

Positional Notation for Natural Numbers in an Arbitrary Base

Charles Staats III

May 26, 2024

Abstract

We demonstrate the existence and uniqueness of the base- n representation of a natural number, where n is any natural number greater than 1. This comes up when trying to translate mathematical contest problems and solutions into Isabelle/HOL.

Contents

1	Infinite sums	1
2	Modular arithmetic	4
3	Digits as sequence	6
4	Little Endian notation	13
5	Big Endian notation	17
6	Exercises	20

theory *DigitsInBase*

imports *HOL-Computational-Algebra.Computational-Algebra HOL-Number-Theory.Number-Theory*

begin

1 Infinite sums

In this section, it is shown that an infinite series *of natural numbers* converges if and only if its terms are eventually zero. Additionally, the notion of a summation starting from an index other than zero is defined. A few obvious lemmas about these notions are established.

definition *eventually-zero* :: $(\text{nat} \Rightarrow \text{-}) \Rightarrow \text{bool}$ **where**
eventually-zero ($D :: \text{nat} \Rightarrow \text{-}$) $\longleftrightarrow (\forall_{\infty} n. D\ n = 0)$

lemma *eventually-zero-char*:

shows *eventually-zero* $D \longleftrightarrow (\exists s. \forall i \geq s. D\ i = 0)$
unfolding *eventually-zero-def*
using *MOST-nat-le* .

There's a lot of commonality between this setup and univariate polynomials, but drawing out the similarities and proving them is beyond the scope of the current version of this theory except for the following lemma.

lemma *eventually-zero-poly*:
shows *eventually-zero* $D \longleftrightarrow D = \text{poly.coeff } (Abs\text{-poly } D)$
by (*metis Abs-poly-inverse MOST-coeff-eq-0 eventually-zero-def mem-Collect-eq*)

lemma *eventually-zero-imp-summable* [*simp*]:
assumes *eventually-zero* D
shows *summable* D
using *summable-finite assms eventually-zero-char*
by (*metis (mono-tags) atMost-iff finite-atMost nat-le-linear*)

lemma *summable-bounded*:
fixes *my-seq* :: $\text{nat} \Rightarrow \text{nat}$ **and** $n :: \text{nat}$
assumes $\bigwedge i. i \geq n \longrightarrow \text{my-seq } i = 0$
shows *summable* *my-seq*
using *assms eventually-zero-char eventually-zero-imp-summable* **by** *blast*

lemma *sum-bounded*:
fixes *my-seq* :: $\text{nat} \Rightarrow \text{nat}$ **and** $n :: \text{nat}$
assumes $\bigwedge i. i \geq n \longrightarrow \text{my-seq } i = 0$
shows $(\sum i. \text{my-seq } i) = (\sum i < n. \text{my-seq } i)$
by (*meson assms finite-lessThan lessThan-iff linorder-not-le suminf-finite*)

lemma *product-seq-eventually-zero*:
fixes *seq1 seq2* :: $\text{nat} \Rightarrow \text{nat}$
assumes *eventually-zero* *seq1*
shows *eventually-zero* $(\lambda i. \text{seq1 } i * \text{seq2 } i)$
using *mult-0 eventually-zero-char*
by (*metis (no-types, lifting) assms*)

abbreviation *upper-sum*
where *upper-sum* $\text{seq } n \equiv \sum i. \text{seq } (i + n)$
syntax
 $\text{-from-sum} :: \text{idt} \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \ ((\exists \sum \text{-}\geq\text{-}./ \text{-}) [0,0,10] 10)$
translations
 $\sum i \geq n. t == \text{CONST } \text{upper-sum } (\lambda i. t) n$

The following two statements are proved as a sanity check. They are not intended to be used anywhere.

corollary
fixes *seq* :: $\text{nat} \Rightarrow \text{nat}$ **and** $a :: \text{nat}$
assumes *seq-def*: $\bigwedge i. \text{seq } i = (\text{if } i = 0 \text{ then } a \text{ else } 0)$
shows $(\sum i \geq 0. \text{seq } i) = \text{upper-sum } (\lambda i. \text{seq } i) 0$

by *simp*

corollary

fixes $seq :: nat \Rightarrow nat$ **and** $a :: nat$

assumes *seq-def*: $\bigwedge i. seq\ i = (if\ i = 0\ then\ a\ else\ 0)$

shows $(\sum_{i \geq 0}. seq\ i) = a$

by (*smt* (*verit*) *group-cancel.rule0 lessI lessThan-0 linorder-not-less seq-def sum.empty sum.lessThan-Suc-shift sum-bounded*)

lemma *bounded-sum-from*:

fixes $seq :: nat \Rightarrow nat$ **and** $n\ s :: nat$

assumes $\forall i > s. seq\ i = 0$ **and** $n \leq s$

shows $(\sum_{i \geq n}. seq\ i) = (\sum_{i=n..s}. seq\ i)$

proof –

have $\bigwedge i. i > (s - n) \implies seq\ (i + n) = 0$

using *assms* **by** (*meson less-diff-conv2*)

then have $(\sum_{i \geq n}. seq\ i) = (\sum_{i \leq s-n}. seq\ (i + n))$

by (*meson atMost-iff finite-atMost leI suminf-finite*)

also have $\dots = (\sum_{i=n..s}. seq\ i)$

proof –

have $\bigwedge na. (\sum_{na \leq na}. seq\ (na + n)) = sum\ seq\ \{0 + n..na + n\}$

by (*metis* (*no-types*) *atLeast0AtMost sum.shift-bounds-cl-nat-ivl*)

then show *?thesis*

by (*simp add: assms(2)*)

qed

finally show *?thesis* .

qed

lemma *split-suminf*:

fixes $seq :: nat \Rightarrow nat$ **and** $n :: nat$

assumes *eventually-zero seq*

shows $(\sum i. seq\ i) = (\sum_{i < n}. seq\ i) + (\sum_{i \geq n}. seq\ i)$

proof –

obtain s **where** $\bigwedge i. i \geq s \longrightarrow seq\ i = 0$

using *assms* **unfolding** *eventually-zero-char* **by** *presburger*

then have *sum-s*: $(\sum i. seq\ i) = (\sum_{i < s}. seq\ i)$

using *sum-bounded* **by** *presburger*

show $(\sum i. seq\ i) = (\sum_{i < n}. seq\ i) + (\sum_{i \geq (n)}. seq\ i)$

proof (*cases* $n \geq s$)

case *True*

then have $(\sum_{i \geq (n)}. seq\ i) = 0$

using s **by** *force*

moreover have $(\sum_{i < n}. seq\ i) = (\sum_{i < s}. seq\ i)$

by (*metis* *True dual-order.trans s sum-bounded sum-s*)

ultimately show *?thesis* **using** *sum-s* **by** *simp*

next

case *False*

have $0: (\sum_{i=n..s}. seq\ i) = (\sum_{i \geq n}. seq\ i)$

by (*metis* *False bounded-sum-from le-eq-less-or-eq nle-le s*)

```

from False have  $n \leq s$ 
  by simp
then have  $(\sum i < s. \text{seq } i) = (\sum i < n. \text{seq } i) + (\sum i = n..s. \text{seq } i)$ 
  by (metis add-cancel-left-right nat-le-iff-add s sum.atLeastLessThan-concat
add-0 lessThan-atLeast0 sum.last-plus)
then show ?thesis using 0 sum-s
  by presburger
qed
qed

```

```

lemma dvd-suminf:
  fixes  $\text{seq} :: \text{nat} \Rightarrow \text{nat}$  and  $b :: \text{nat}$ 
  assumes eventually-zero seq and  $\bigwedge i. b \text{ dvd } \text{seq } i$ 
  shows  $b \text{ dvd } (\sum i. \text{seq } i)$ 
proof –
  obtain  $s :: \text{nat}$  where  $s: i \geq s \implies \text{seq } i = 0$  for  $i$ 
  using assms(1) eventually-zero-char by blast
  then have  $(\sum i. \text{seq } i) = (\sum i < s. \text{seq } i)$ 
  using sum-bounded by blast
  moreover have  $b \text{ dvd } (\sum i < s. \text{seq } i)$ 
  using assms(2) by (simp add: dvd-sum)
  ultimately show ?thesis by presburger
qed

```

```

lemma eventually-zero-shifted:
  assumes eventually-zero seq
  shows eventually-zero  $(\lambda i. \text{seq } (i + n))$ 
  by (meson assms eventually-zero-char trans-le-add1)

```

2 Modular arithmetic

This section establishes a number of lemmas about modular arithmetic, including that modular division distributes over an “infinite” sum whose terms are eventually zero.

```

lemma pmod-int-char:
  fixes  $a b d :: \text{int}$ 
  shows  $[a = b] \text{ (mod } d) \longleftrightarrow (\exists (n :: \text{int}). a = b + n * d)$ 
  by (metis cong-iff-lin cong-sym mult.commute)

```

```

lemma equiv-conj-if:
  assumes  $P \implies Q$  and  $P \implies R$  and  $Q \implies R \implies P$ 
  shows  $P \longleftrightarrow Q \wedge R$ 
  using assms by auto

```

```

lemma mod-successor-char:
  fixes  $k k' i b :: \text{nat}$ 
  assumes  $(b :: \text{nat}) \geq 2$ 

```

shows $[k = k'] \pmod{b^{\wedge}(Suc\ i)} \longleftrightarrow [k\ div\ b^{\wedge}i = k'\ div\ b^{\wedge}i] \pmod{b} \wedge [k = k'] \pmod{b^{\wedge}i}$
proof (*rule equiv-conj-if*)
assume $kk'-cong: [k = k'] \pmod{b^{\wedge}Suc\ i}$
then show $[k\ div\ b^{\wedge}i = k'\ div\ b^{\wedge}i] \pmod{b}$
by (*smt (verit, ccfv-SIG) Groups.mult-ac(2) add-diff-cancel-right' cong-def div-mult-mod-eq mod-mult2-eq mod-mult-self4 mult-cancel1 power-Suc2*)
from $kk'-cong$ **show** $[k = k'] \pmod{b^{\wedge}i}$
using *Cong.cong-dvd-modulus-nat*
by (*meson Suc-n-not-le-n le-imp-power-dvd nat-le-linear*)
next
assume $[k\ div\ b^{\wedge}i = k'\ div\ b^{\wedge}i] \pmod{b}$
moreover assume $[k = k'] \pmod{b^{\wedge}i}$
ultimately show $[k = k'] \pmod{b^{\wedge}Suc\ i}$
by (*metis (mono-tags, lifting) cong-def mod-mult2-eq power-Suc2*)
qed

lemma mod-1:
fixes $k\ k'\ b :: nat$
shows $[k = k'] \pmod{b^{\wedge}0}$
by *simp*

lemma mod-distributes:
fixes $seq :: nat \Rightarrow nat$ **and** $b :: nat$
assumes $\exists n. \forall i \geq n. seq\ i = 0$
shows $[(\sum i. seq\ i) = (\sum i. seq\ i\ mod\ b)] \pmod{b}$
proof –
obtain n **where** $n: \bigwedge i. i \geq n \longrightarrow seq\ i = 0$
using *assms by presburger*
from n **have** $(\sum i. seq\ i) = (\sum i < n. seq\ i)$
using *sum-bounded by presburger*
moreover from n **have** $(\sum i. seq\ i\ mod\ b) = (\sum i < n. seq\ i\ mod\ b)$
using *sum-bounded by presburger*
ultimately show *?thesis*
unfolding *cong-def*
by (*metis mod-sum-eq*)
qed

lemma another-mod-cancellation-int:
fixes $a\ b\ c\ d\ m :: int$
assumes $d > 0$ **and** $[m = a + b] \pmod{c * d}$ **and** $a\ div\ d = 0$ **and** $d\ dvd\ b$
shows $[m\ div\ d = b\ div\ d] \pmod{c}$
proof (*subst pmod-int-char*)
obtain $k :: int$ **where** $k: m = a + b + k * c * d$
using *pmod-int-char assms(2) by (metis mult.assoc)*
have $d\ dvd\ (b + k * c * d)$ **using** $\langle d\ dvd\ b \rangle$
by *simp*
from k **have** $m\ div\ d = (a + b + k * c * d)\ div\ d$

```

    by presburger
  also have ... = (b + k*c*d) div d
    using ⟨a div d = 0⟩ ⟨d dvd (b + k*c*d)⟩
    by fastforce
  also have ... = (b div d) + k*c
    using ⟨d dvd b⟩ ⟨d > 0⟩ by auto
  finally show ∃ n. m div d = b div d + n * c
    by blast
qed

```

lemma *another-mod-cancellation*:

```

  fixes a b c d m :: nat
  assumes d > 0 and [m = a + b] (mod c * d) and a div d = 0 and d dvd b
  shows [m div d = b div d] (mod c)
  by (smt (verit) another-mod-cancellation-int assms cong-int-iff of-nat-0 of-nat-0-less-iff
      of-nat-add of-nat-dvd-iff of-nat-mult zdiv-int)

```

3 Digits as sequence

Rules are introduced for computing the i^{th} digit of a base- b representation and the number of digits required to represent a given number. (The latter is essentially an integer version of the base- b logarithm.) It is shown that the sum of the terms $d_i b^i$ converges to m if d_i is the i^{th} digit m . It is shown that the sequence of digits defined is the unique sequence of digits less than b with this property

Additionally, the `digits_in_base` locale is introduced, which specifies a single symbol b referring to a natural number greater than one (the base of the digits). Consequently this symbol is omitted from many of the following lemmas and definitions.

```

locale digits-in-base =
  fixes b :: nat
  assumes b-gte-2: b ≥ 2
begin

```

lemma *b-facts* [*simp*]:

```

  shows b > 1 and b > 0 and b ≠ 1 and b ≠ 0 and 1 mod b = 1 and 1 div b
  = 0
  using b-gte-2 by force+

```

Definition based on [1].

abbreviation *ith-digit* :: nat ⇒ nat ⇒ nat **where**
ith-digit m i ≡ (m div bⁱ) mod b

lemma *ith-digit-lt-base*:

```

  fixes m i :: nat
  shows 0 ≤ ith-digit m i and ith-digit m i < b
  apply (rule Nat.le0)

```

```

using b-facts(2) mod-less-divisor by presburger

definition num-digits :: nat ⇒ nat
  where num-digits m = (LEAST i. m < bi)

lemma num-digits-works:
  shows m < b(num-digits m)
  by (metis LeastI One-nat-def b-facts(1) num-digits-def power-gt-expt)

lemma num-digits-le:
  assumes m < bi
  shows num-digits m ≤ i
  using assms num-digits-works[of m] Least-le num-digits-def
  by metis

lemma num-digits-zero:
  fixes m :: nat
  assumes num-digits m = 0
  shows m = 0
  using num-digits-works[of m]
  unfolding assms
  by simp

lemma num-digits-gt:
  assumes m ≥ bi
  shows num-digits m > i
  by (meson assms b-facts(2) dual-order.strict-trans2 nat-power-less-imp-less num-digits-works)

lemma num-digits-eqI [intro]:
  assumes m ≥ bi and m < b(i+1)
  shows num-digits m = i + 1
proof –
  {
    fix j :: nat
    assume j < i + 1
    then have m ≥ bj
    by (metis Suc-eq-plus1 assms(1) b-facts(1) less-Suc-eq-le order-trans power-increasing-iff)
  }
  then show ?thesis
  using num-digits-works
  unfolding num-digits-def
  by (meson assms(2) leD linorder-neqE-nat not-less-Least)
qed

lemma num-digits-char:
  assumes m ≥ 1
  shows num-digits m = i + 1 ⇔ m ≥ bi ∧ m < b(i+1)
  by (metis add-diff-cancel-right' assms b-gte-2 ex-power-ivl1 num-digits-eqI)

```

Statement based on [1].

```

lemma num-digits-recurrence:
  fixes m :: nat
  assumes m ≥ 1
  shows num-digits m = num-digits (m div b) + 1
proof –
  define nd where nd = num-digits m
  then have lb: m ≥ b(nd-1) and ub: m < bnd
    using num-digits-char[OF assms]
    apply (metis assms diff-is-0-eq le-add-diff-inverse2 nat-le-linear power-0)
    using nd-def num-digits-works by presburger
  from ub have ub2: m div b < b(nd-1)
    by (metis Suc-eq-plus1 add commute add-diff-inverse-nat assms less-mult-imp-div-less
less-one
linorder-not-less mult.commute power.simps(2) power-0)
  from lb have lb2: m div b ≥ b(nd - 1) div b
    using div-le-mono by presburger
  show ?thesis
  proof (cases m ≥ b)
    assume m ≥ b
    then have nd ≥ 2
      unfolding nd-def
      by (metis One-nat-def assms less-2-cases-iff linorder-not-le nd-def power-0
power-one-right
ub)
    then have m div b ≥ b(nd-2)
      using lb2
    by (smt (verit) One-nat-def add-le-imp-le-diff b-facts(4) diff-diff-left le-add-diff-inverse2
nonzero-mult-div-cancel-left numeral-2-eq-2 plus-1-eq-Suc power-add power-commutes
power-one-right)
    then show ?thesis
      using ub2 num-digits-char assms nd-def
      by (smt (verit) ⟨2 ≤ nd⟩ add-diff-cancel-right' add-leD2 add-le-imp-le-diff
diff-diff-left
eq-diff-iff le-add2 nat-1-add-1 num-digits-eqI)
  next
    assume ¬ b ≤ m
    then have m < b
      by simp
    then have num-digits m = 1
      using assms
      by (metis One-nat-def Suc-eq-plus1 num-digits-char power-0 power-one-right)
    from ⟨m < b⟩ have m div b = 0
      using div-less by presburger
    then have num-digits (m div b) = 0
      using Least-eq-0 num-digits-def by presburger
    show ?thesis
      using ⟨num-digits (m div b) = 0⟩ ⟨num-digits m = 1⟩ by presburger
  qed
qed

```



```

lemma num-digits-zero-2 [simp]: num-digits 0 = 0
  by (simp add: num-digits-def)

end

locale base-10
begin

  As a sanity check, the number of digits in base ten is computed for several
  natural numbers.

sublocale digits-in-base 10
  by (unfold-locales, simp)

corollary
  shows num-digits 0 = 0
    and num-digits 1 = 1
    and num-digits 9 = 1
    and num-digits 10 = 2
    and num-digits 99 = 2
    and num-digits 100 = 3
  by (simp-all add: num-digits-recurrence)

end

context digits-in-base
begin

lemma high-digits-zero-helper:
  fixes m i :: nat
  shows i < num-digits m ∨ ith-digit m i = 0
proof (cases i < num-digits m)
  case True
  then show ?thesis by meson
next
  case False
  then have i ≥ num-digits m by force
  then have m < bi
  by (meson b-facts(1) num-digits-works order-less-le-trans power-increasing-iff)
  then show ?thesis
  by simp
qed

lemma high-digits-zero:
  fixes m i :: nat
  assumes i ≥ num-digits m
  shows ith-digit m i = 0
  using high-digits-zero-helper assms leD by blast

```

lemma *digit-expansion-bound*:
fixes $i :: \text{nat}$ **and** $A :: \text{nat} \Rightarrow \text{nat}$
assumes $\bigwedge j. A\ j < b$
shows $(\sum_{j < i}. A\ j * b^{\wedge}j) < b^{\wedge}i$
proof (*induct i*)
case (*Suc i*)
show ?*case*
proof (*subst Set-Interval.comm-monoid-add-class.sum.lessThan-Suc*)
have $A\ i * b^{\wedge}i \leq (b-1) * b^{\wedge}i$ **using** *assms*
by (*metis One-nat-def Suc-pred b-facts(2) le-simps(2) mult-le-mono1*)
then have $(\sum_{j < i}. A\ j * b^{\wedge}j) + A\ i * b^{\wedge}i < b^{\wedge}i + (b-1) * b^{\wedge}i$
using *Suc add-less-le-mono* **by** *blast*
also have $\dots \leq b^{\wedge} \text{Suc } i$
using *assms(1) mult-eq-if* **by** *auto*
finally show $(\sum_{j < i}. A\ j * b^{\wedge}j) + A\ i * b^{\wedge}i < b^{\wedge} \text{Suc } i$.
qed
qed *simp*

Statement and proof based on [1].

lemma *num-digits-suc*:
fixes $n\ m :: \text{nat}$
assumes $\text{Suc } n = \text{num-digits } m$
shows $n = \text{num-digits } (m \text{ div } b)$
using *num-digits-recurrence assms*
by (*metis One-nat-def Suc-eq-plus1 Suc-le-lessD le-add2 linorder-not-less num-digits-le old.nat.inject power-0*)

Proof (and to some extent statement) based on [1].

lemma *digit-expansion-bounded-seq*:
fixes $m :: \text{nat}$
shows $m = (\sum_{j < \text{num-digits } m}. \text{ith-digit } m\ j * b^{\wedge}j)$
proof (*induct num-digits m arbitrary: m*)
case 0
then show $m = (\sum_{j < \text{num-digits } m}. \text{ith-digit } m\ j * b^{\wedge}j)$
using *lessThan-0 sum.empty num-digits-zero* **by** *metis*
next
case (*Suc nd m*)
assume $nd: \text{Suc } nd = \text{num-digits } m$
define c **where** $c = m \text{ mod } b$
then have $m \text{ exp } b = b * (m \text{ div } b) + c$ **and** $c < b$
by *force+*
show $m = (\sum_{j < \text{num-digits } m}. \text{ith-digit } m\ j * b^{\wedge}j)$
proof –
have $nd = \text{num-digits } (m \text{ div } b)$
using *num-digits-suc[OF nd]*.
with *Suc* **have** $m \text{ div } b = (\sum_{j < nd}. \text{ith-digit } (m \text{ div } b)\ j * b^{\wedge}j)$
by *presburger*
with *mexp* **have** $m = b * (\sum_{j < nd}. \text{ith-digit } (m \text{ div } b)\ j * b^{\wedge}j) + c$
by *presburger*

also have ... = $(\sum j < nd. \text{ith-digit } (m \text{ div } b) j * b^{\widehat{\text{Suc } j}}) + c$
by (*simp add: sum-distrib-left mult.assoc mult.commute*)
also have ... = $(\sum j < nd. \text{ith-digit } m (\text{Suc } j) * b^{\widehat{\text{Suc } j}}) + c$
by (*simp add: div-mult2-eq*)
also have ... = $(\sum j = \text{Suc } 0 .. < \text{Suc } nd. \text{ith-digit } m j * b^{\widehat{j}}) + \text{ith-digit } m 0$
unfolding *sum.shift-bounds-Suc-ivl c-def atLeast0LessThan*
by *simp*
also have ... = $(\sum j < \text{Suc } nd. \text{ith-digit } m j * b^{\widehat{j}})$
by (*smt (verit) One-nat-def Zero-not-Suc add.commute add-diff-cancel-right'*
atLeast0LessThan
calculation div-by-Suc-0 mult.commute nonzero-mult-div-cancel-left power-0
sum.lessThan-Suc-shift sum.shift-bounds-Suc-ivl)
also note *nd*
finally show $m = (\sum j < \text{num-digits } m. \text{ith-digit } m j * b^{\widehat{j}})$.
qed
qed

A natural number can be obtained from knowing all its base- b digits by the formula $\sum_j d_j b^j$.

theorem *digit-expansion-seq:*

fixes $m :: \text{nat}$

shows $m = (\sum j. \text{ith-digit } m j * b^{\widehat{j}})$

using *digit-expansion-bounded-seq[of m] high-digits-zero[of m] sum-bounded mult-0*

by (*metis (no-types, lifting)*)

lemma *lower-terms:*

fixes $a c i :: \text{nat}$

assumes $c < b^{\widehat{i}}$ **and** $a < b$

shows $\text{ith-digit } (a * b^{\widehat{i}} + c) i = a$

using *assms* **by** *force*

lemma *upper-terms:*

fixes $A a i :: \text{nat}$

assumes $b * b^{\widehat{i}} \text{ dvd } A$ **and** $a < b$

shows $\text{ith-digit } (A + a * b^{\widehat{i}}) i = a$

using *assms* **by** *force*

lemma *current-term:*

fixes $A a c i :: \text{nat}$

assumes $b * b^{\widehat{i}} \text{ dvd } A$ **and** $c < b^{\widehat{i}}$ **and** $a < b$

shows $\text{ith-digit } (A + a * b^{\widehat{i}} + c) i = a$

proof –

have $(A + a * b^{\widehat{i}} + c) \text{ div } b^{\widehat{i}} \text{ mod } b = (a * b^{\widehat{i}} + c) \text{ div } b^{\widehat{i}} \text{ mod } b$

using *assms(1)*

by (*metis (no-types, lifting) div-eq-0-iff add-cancel-right-right*

assms(2) assms(3) div-plus-div-distrib-dvd-left dvd-add-times-triv-right-iff

dvd-mult-right lower-terms upper-terms)

also have ... = a

using *assms* **by** *force*

finally show $(A + a * b^i + c) \text{ div } b^i \text{ mod } b = a$.
qed

Given that

$$m = \sum_i d_i b^i$$

where the d_i are all natural numbers less than b , it follows that d_j is the j^{th} digit of m .

theorem *seq-uniqueness*:

fixes $m\ j :: \text{nat}$ **and** $D :: \text{nat} \Rightarrow \text{nat}$

assumes *eventually-zero* D **and** $m = (\sum i. D\ i * b^i)$ **and** $\bigwedge i. D\ i < b$

shows $D\ j = \text{ith-digit}\ m\ j$

proof –

have *eventually-zero* (*ith-digit* m)

using *high-digits-zero*

by (*meson eventually-zero-char*)

then have *term-eventually-zero: eventually-zero* $(\lambda i. D\ i * b^i)$

using *product-seq-eventually-zero* *assms(1)* **by** *auto*

then have *shifted-term-eventually-zero*:

eventually-zero $(\lambda i. D\ (i + n) * b^{i + n})$ **for** n

using *eventually-zero-shifted*

by *blast*

note $\langle m = (\sum i. D\ i * b^i) \rangle$

then have *two-sums*: $m = (\sum i < \text{Suc}\ j. D\ i * b^i) + (\sum i \geq \text{Suc}\ j. D\ i * b^i)$

using *split-suminf[OF term-eventually-zero]* **by** *presburger*

have $i \geq \text{Suc}\ j \implies b * b^j \text{ dvd } (D\ i * b^i)$ **for** i

by (*metis dvd-mult2 le-imp-power-dvd mult.commute power-Suc*)

then have $b * b^j \text{ dvd } (\sum i \geq \text{Suc}\ j. D\ i * b^i)$

using *dvd-suminf shifted-term-eventually-zero le-add2*

by *presburger*

with *two-sums* **have** $[m = (\sum i < \text{Suc}\ j. D\ i * b^i)] \text{ (mod } b * b^j)$

by (*meson cong-def Cong.cong-dvd-modulus-nat mod-add-self2*)

then have *one-sum*: $[m = (\sum i < j. D\ i * b^i) + D\ j * b^j] \text{ (mod } b * b^j)$

by *simp*

have $(\sum i < j. D\ i * b^i) < b^j$

using *assms(3) digit-expansion-bound* **by** *blast*

with *one-sum* **have** $[m \text{ div } b^j = (D\ j)] \text{ (mod } b)$

using *another-mod-cancellation dual-order.strict-trans1*

unfolding *cong-def*

by *auto*

then show $D\ j = \text{ith-digit}\ m\ j$

using *assms(3) mod-less unfolding cong-def* **by** *presburger*

qed

end

4 Little Endian notation

In this section we begin to define finite digit expansions. Ultimately we want to write digit expansions in “big endian” notation, by which we mean with the highest place digit on the left and the ones digit on the right, since this ordering is standard in informal mathematics. However, it is easier to first define “little endian” expansions with the ones digit on the left since that way the list indexing agrees with the sequence indexing used in the previous section (whenever both are defined).

Notation, definitions, and lemmas in this section typically start with the prefix **LE** (for “little endian”) to distinguish them from the big endian versions in the next section.

```
fun LEeval-as-base (-LEbase - [65, 65] 70)
  where [] LEbase b = 0
  | (d # d-list) LEbase b = d + b * (d-list LEbase b)
```

```
corollary shows [2, 4] LEbase 5 = (22::nat)
by simp
```

```
lemma LEbase-one-digit [simp]: shows [a::nat] LEbase b = a
by simp
```

```
lemma LEbase-two-digits [simp]: shows [a0::nat, a1] LEbase b = a0 + a1 * b
by simp
```

```
lemma LEbase-three-digits [simp]: shows [a0::nat, a1, a2] LEbase b = a0 + a1*b
+ a2*b^2
```

```
proof -
```

```
  have [a0::nat, a1, a2] LEbase b = a0 + ([a1, a2] LEbase b) * b
  by simp
```

```
  also have ... = a0 + (a1 + a2*b) * b
  by simp
```

```
  also have ... = a0 + a1*b + a2*b^2
  by (simp add: add-mult-distrib power2-eq-square)
```

```
  finally show ?thesis .
```

```
qed
```

```
lemma LEbase-closed-form:
```

```
shows (A :: nat list) LEbase b = (∑ i < length A . A!i * b^i)
```

```
proof (induct A)
```

```
  case Nil
```

```
  show ?case
```

```
  by simp
```

```
next
```

```
  case (Cons a A)
```

```
  show ?case
```

```
  proof -
```

```

have (a # A)LEbase b = a + b * (ALEbase b)
  by simp
also have ... = a + b * (∑ i < length A. A!i * b ^ i)
  using Cons by simp
also have ... = a + (∑ i < length A. b * A!i * b ^ i)
  by (smt (verit) mult.assoc sum.cong sum-distrib-left)
also have ... = a + (∑ i < length A. A!i * b ^ (i+1))
  by (simp add: mult.assoc mult.left-commute)
also have ... = a + (∑ i < length A. (a#A)!(i+1) * b ^ (i+1))
  by force
also have ... = (a#A)!0 * b ^ 0 + (∑ i < length A. (a#A)!(Suc i) * b ^ (Suc i))
  by force
also have ... = (∑ i < length (a # A). (a#A)!i * b ^ i)
  using sum.lessThan-Suc-shift
  by (smt (verit) length-Cons sum.cong)
finally show ?thesis .
qed
qed

```

lemma *LEbase-concatenate:*

```

fixes A D :: nat list and b :: nat
shows (A @ D)LEbase b = (ALEbase b) + b ^ (length A) * (DLEbase b)
proof (induct A)
  case Nil
  show ?case
    by simp
next
  case (Cons a A)
  show ?case
  proof -
    have ((a # A) @ D)LEbase b = ((a # (A @ D))LEbase b)
      by simp
    also have ... = a + b * ((A @ D)LEbase b)
      by simp
    also have ... = a + b * (ALEbase b + b ^ length A * (DLEbase b))
      unfolding Cons by rule
    also have ... = (a + b * (ALEbase b)) + b ^ (length (a#A)) * (DLEbase b)
      by (simp add: distrib-left)
    also have ... = ((a # A)LEbase b) + b ^ length (a # A) * (DLEbase b)
      by simp
    finally show ?thesis .
  qed
qed

```

context *digits-in-base*

begin

definition *LEdigits* :: nat ⇒ nat list **where**

LEdigits m = [ith-digit m i. i ← [0..<(num-digits m)]]

lemma *length-is-num-digits*:
fixes $m :: nat$
shows $length (LEdigits\ m) = num-digits\ m$
unfolding *LEdigits-def* **by** *simp*

lemma *ith-list-element* [*simp*]:
assumes $(i::nat) < length (LEdigits\ m)$
shows $(LEdigits\ m) ! i = ith-digit\ m\ i$
using *assms*
by (*simp add: length-is-num-digits LEdigits-def*)

lemma *LEbase-infinite-sum*:
fixes $m :: nat$
shows $(\sum i. ith-digit\ m\ i * b^i) = (LEdigits\ m)_{LEbase\ b}$
proof (*unfold LEdigits-def LEbase-closed-form*)
have
 $(\sum i < length (map (ith-digit\ m) [0..<num-digits\ m])).$
 $map (ith-digit\ m) [0..<num-digits\ m] ! i *$
 b^i
 $= (\sum i < num-digits\ m. map (ith-digit\ m) [0..<num-digits\ m] ! i * b^i)$
using *LEdigits-def length-is-num-digits* **by** *presburger*
also have $\dots = (\sum i < num-digits\ m. ith-digit\ m\ i * b^i)$
by *force*
also have $\dots = (\sum i. ith-digit\ m\ i * b^i)$
using *sum-bounded high-digits-zero mult-0*
by (*metis (no-types, lifting)*)
finally show
 $(\sum i. ith-digit\ m\ i * b^i) =$
 $(\sum i < length (map (ith-digit\ m) [0..<num-digits\ m])). map (ith-digit\ m) [0..<num-digits$
 $m] ! i * b^i$
by *presburger*
qed

lemma *digit-expansion-LElist*:
fixes $m :: nat$
shows $(LEdigits\ m)_{LEbase\ b} = m$
using *digit-expansion-seq LEbase-infinite-sum*
by *presburger*

lemma *LElist-uniqueness*:
fixes $D :: nat\ list$
assumes $\forall i < length\ D. D!i < b$ **and** $D = [] \vee last\ D \neq 0$
shows $LEdigits (D_{LEbase\ b}) = D$
proof –
define *seq* **where** *seq* $i = (if\ i < length\ D\ then\ D!i\ else\ 0)$ **for** i
then have *seq-bound*: $i \geq length\ D \implies seq\ i = 0$ **for** i
by *simp*
then have *seq-eventually-zero*: *eventually-zero seq*

```

    using eventually-zero-char by blast
    have ith-digit-connection:  $i < \text{num-digits } m \implies (\text{LEdigits } m)!i = \text{ith-digit } m \ i$ 
  for  $m \ i$ 
    unfolding LEdigits-def by simp
    let  $?m = D_{LEbase} \ b$ 
    have length-bounded-sum:  $D_{LEbase} \ b = (\sum_{i < \text{length } D} \text{seq } i * b^i)$ 
    unfolding LEbase-closed-form seq-def by force
    also have  $\dots = (\sum i. \text{seq } i * b^i)$ 
    using seq-bound sum-bounded by fastforce
    finally have seq-is-digits:  $\text{seq } j = \text{ith-digit } ?m \ j$  for  $j$ 
    using seq-uniqueness[OF seq-eventually-zero] assms(1)
    by (metis b-facts(2) seq-def)
    then have  $i < \text{length } D \implies \text{ith-digit } ?m \ i = D!i$  for  $i$ 
    using seq-def by presburger
    then have  $i < \text{length } D \implies i < \text{num-digits } ?m \implies (\text{LEdigits } ?m)!i = D!i$  for  $i$ 
    using ith-digit-connection[of  $i \ ?m$ ] by presburger
    moreover have  $\text{length } D = \text{num-digits } ?m$ 
    proof (rule le-antisym)
      show  $\text{length } D \leq \text{num-digits } ?m$ 
      proof (cases  $D = []$ )
        assume  $D \neq []$ 
        then have  $\text{last } D \neq 0$  using assms(2) by auto
        then have  $\text{last } D \geq 1$  by simp
        have  $?m \geq \text{seq } (\text{length } D - 1) * b^{(\text{length } D - 1)}$ 
        using length-bounded-sum
        by (metis b-facts(2) less-eq-div-iff-mult-less-eq mod-less-eq-dividend seq-is-digits
        zero-less-power)
        then have  $?m \geq (\text{last } D) * b^{(\text{length } D - 1)}$ 
        by (simp add:  $\langle D \neq [] \rangle$  last-conv-nth seq-def)
        with  $\langle \text{last } D \geq 1 \rangle$  have  $?m \geq b^{(\text{length } D - 1)}$ 
        by (metis le-trans mult-1 mult-le-mono1)
        then show  $\text{num-digits } ?m \geq \text{length } D$ 
        using num-digits-gt not-less-eq
        by (metis One-nat-def Suc-pred  $\langle D \neq [] \rangle$  bot-nat-0.extremum-uniqueI leI
        length-0-conv)
      qed simp
      show  $\text{num-digits } ?m \leq \text{length } D$ 
      by (metis length-bounded-sum seq-is-digits digit-expansion-bound ith-digit-lt-base(2)
        num-digits-le)
    qed
  qed
  ultimately show ?thesis
  by (simp add: length-is-num-digits list-eq-iff-nth-eq)
qed

lemma LE-digits-zero [simp]:  $\text{LEdigits } 0 = []$ 
  using LEdigits-def by auto

lemma LE-units-digit [simp]:
  assumes  $(m::\text{nat}) \in \{1..<b\}$ 

```


shows $LEdigits\ m = [m]$
using *assms* $LEdigits-def\ num-digits-recurrence$ **by** *auto*

end

5 Big Endian notation

In this section the desired representation of natural numbers, as finite lists of digits with the highest place on the left, is finally realized.

definition $BEeval-as-base\ (-base\ -\ [65,\ 65]\ 70)$
where $[simp]: D_{base\ b} = (rev\ D)_{LEbase\ b}$

corollary shows $[4,\ 2]_{base\ 5} = (22::nat)$
by *simp*

lemma $BEbase-one-digit\ [simp]:$ **shows** $[a::nat]_{base\ b} = a$
by *simp*

lemma $BEbase-two-digits\ [simp]:$ **shows** $[a_1::nat,\ a_0]_{base\ b} = a_1*b + a_0$
by *simp*

lemma $BEbase-three-digits\ [simp]:$ **shows** $[a_2::nat,\ a_1,\ a_0]_{base\ b} = a_2*b^2 + a_1*b + a_0$

proof –

have $b * (a_1 + b * a_2) = a_2 * b^2 + a_1 * b$
apply (*subst mult.commute*)
unfolding *add-mult-distrib power2-eq-square*
by *simp*

then show *?thesis* **by** *simp*

qed

lemma $BEbase-closed-form:$

fixes $A :: nat\ list$ **and** $b :: nat$

shows $A_{base\ b} = (\sum\ i < length\ A.\ A[i] * b^{(length\ A - Suc\ i)})$

unfolding *LEbase-closed-form BEeval-as-base-def*

apply (*subst sum.nat-diff-reindex[symmetric]*)

apply (*subst length-rev*)

using *rev-nth*

by (*metis (no-types, lifting) length-rev lessThan-iff rev-rev-ident sum.cong*)

lemma $BEbase-concatenate:$

fixes $A\ D :: nat\ list$ **and** $b :: nat$

shows $(A @ D)_{base\ b} = (A_{base\ b}) * b^{(length\ D)} + (D_{base\ b})$

using *LEbase-concatenate* **by** *simp*

context *digits-in-base*

begin

definition $digits :: nat \Rightarrow nat\ list$ **where**
 $digits\ m = rev\ (LEdigits\ m)$

lemma *length-is-num-digits-2*:
fixes $m :: nat$
shows $length\ (digits\ m) = num-digits\ m$
using *length-is-num-digits digits-def* **by** *simp*

lemma *LE-BE-equivalence*:
fixes $m :: nat$
shows $(digits\ m)_{base\ b} = (LEdigits\ m)_{LEbase\ b}$
by (*simp add: digits-def*)

lemma *BEbase-infinite-sum*:
fixes $m :: nat$
shows $(\sum i. ith-digit\ m\ i * b^i) = (digits\ m)_{base\ b}$
using *LE-BE-equivalence LEbase-infinite-sum* **by** *presburger*

Every natural number can be represented in base b , specifically by the digits sequence defined earlier.

theorem *digit-expansion-list*:
fixes $m :: nat$
shows $(digits\ m)_{base\ b} = m$
using *LE-BE-equivalence digit-expansion-LElist* **by** *auto*

If two natural numbers have the same base- b representation, then they are equal.

lemma *digits-cancellation*:
fixes $k\ m :: nat$
assumes $digits\ k = digits\ m$
shows $k = m$
by (*metis assms digit-expansion-list*)

Suppose we have a finite (possibly empty) sequence D_1, \dots, D_n of natural numbers such that $0 \leq D_i < b$ for all i and such that D_1 , if it exists, is nonzero. Then this sequence is the base- b representation for $\sum_i D_i b^{n-i}$.

theorem *list-uniqueness*:
fixes $D :: nat\ list$
assumes $\forall d \in set\ D. d < b$ **and** $D = [] \vee D!0 \neq 0$
shows $digits\ (D_{base\ b}) = D$
unfolding *digits-def BEeval-as-base-def*
using *LElist-uniqueness*
by (*metis Nil-is-rev-conv One-nat-def assms last-conv-nth length-greater-0-conv nth-mem rev-nth rev-swap set-rev*)

We now prove some simplification rules (including a recurrence relation) to make it easier for Isabelle/HOL to compute the base- b representation of a natural number.

The base- b representation of 0 is empty, at least following the conventions of this theory file.

lemma *digits-zero* [*simp*]:
shows $\text{digits } 0 = []$
by (*simp add: digits-def*)

If $0 < m < b$, then the base- b representation of m consists of a single digit, namely m itself.

lemma *single-digit-number* [*simp*]:
assumes $m \in \{0 < .. < b\}$
shows $\text{digits } m = [m]$
using *assms digits-def* **by** *auto*

For all $m \geq b$, the base- b representation of m consists of the base- b representation of $[m/b]$ followed by (as the last digit) the remainder of m when divided by b .

lemma *digits-recurrence* [*simp*]:
assumes $m \geq b$
shows $\text{digits } m = (\text{digits } (m \text{ div } b)) @ [m \text{ mod } b]$

proof –

have *num-digits* $m > 1$
using *assms* **by** (*simp add: num-digits-gt*)
then have *num-digits* $m > 0$
by *simp*
then have *num-digits* $(m \text{ div } b) = \text{num-digits } m - 1$
by (*metis Suc-diff-1 num-digits-suc*)
have $k > 0 \implies \text{last } (\text{rev } [0..<k]) = 0$ **for** $k::\text{nat}$
by (*simp add: last-rev*)
have $[\text{Suc } 0..<\text{Suc } k] = [\text{Suc } i. i \leftarrow [0..<k]]$ **for** $k::\text{nat}$
using *map-Suc-upt* **by** *presburger*
then have $\text{rev } [\text{Suc } 0..<\text{Suc } k] = [\text{Suc } i. i \leftarrow \text{rev } [0..<k]]$ **for** $k::\text{nat}$
by (*metis rev-map*)
then have $[f i. i \leftarrow \text{rev } [\text{Suc } 0..<\text{Suc } k]] = [f (\text{Suc } i). i \leftarrow \text{rev } [0..<k]]$ **for** f
and $k::\text{nat}$
by *simp*
then have *map-shift*: $k > 0 \implies [f i. i \leftarrow \text{rev } [1..<k]] = [f (\text{Suc } i). i \leftarrow \text{rev } [0..<(k-1)]]$
for f **and** $k::\text{nat}$
by (*metis One-nat-def Suc-diff-1*)
have *digit-down*: $\text{ith-digit } m (\text{Suc } i) = \text{ith-digit } (m \text{ div } b) i$ **for** $i::\text{nat}$
by (*simp add: div-mult2-eq*)
have $\text{digits } m = \text{rev } [\text{ith-digit } m i. i \leftarrow [0..<\text{num-digits } m]]$
using *LEdigits-def digits-def* **by** *presburger*
also have $\dots = [\text{ith-digit } m i. i \leftarrow \text{rev } [0..<\text{num-digits } m]]$
using *rev-map* **by** *blast*
also have $\dots = [\text{ith-digit } m i. i \leftarrow \text{butlast } (\text{rev } [0..<\text{num-digits } m])] @$
 $[\text{ith-digit } m (\text{last } (\text{rev } [0..<\text{num-digits } m]))]$
by (*metis (no-types, lifting) Nil-is-map-conv Nil-is-rev-conv <1 < num-digits m*)

```

    bot-nat-0.extremum-strict dual-order.strict-trans1 last-map map-butlast
snoc-eq-iff-butlast
    upt-eq-Nil-conv)
also have ... = [ith-digit m i. i ← rev [1..<num-digits m]] @
    [ith-digit m 0]
    using ⟨1 < num-digits m⟩ ⟨∧k. 0 < k ⇒ last (rev [0..<k]) = 0⟩ by fastforce
also have ... = [ith-digit m (Suc i). i ← rev [0..<(num-digits m - 1)]] @
    [ith-digit m 0]
    using map-shift[OF ⟨num-digits m > 0⟩] by blast
also have ... = [ith-digit (m div b) i. i ← rev [0..<(num-digits m - 1)]] @
    [ith-digit m 0]
    using digit-down by presburger
also have ... = (digits (m div b)) @ [ith-digit m 0]
    by (simp add: LEdigits-def ⟨num-digits (m div b) = num-digits m - 1⟩ digits-def
rev-map)
also have ... = (digits (m div b)) @ [m mod b]
    by simp
finally show ?thesis .
qed

end

```

6 Exercises

This section contains demonstrations of how to denote certain facts with the notation of the previous sections, and how to quickly prove those facts using the lemmas and theorems above.

The base-5 representation of 22 is 42₅.

```

corollary digits-in-base.digits 5 22 = [4, 2]
proof –
    interpret digits-in-base 5
    by (simp add: digits-in-base.intro)
    show digits 22 = [4, 2]
    by simp
qed

```

A different proof of the same statement.

```

corollary digits-in-base.digits 5 22 = [4, 2]
proof –
    interpret digits-in-base 5
    by (simp add: digits-in-base.intro)
    have [4, 2]base 5 = (22::nat)
    by simp
    have d ∈ set [4, 2] ⇒ d < 5 for d::nat
    by fastforce
    then show ?thesis
    using list-uniqueness

```

by (*metis* $\langle [4, 2]_{base\ 5} = 22 \rangle$ *nth-Cons-0 numeral-2-eq-2 zero-neq-numeral*)
qed

end

References

- [1] B. Porter. Threedivides. <https://isabelle.in.tum.de/dist/library/HOL/HOL-ex/ThreeDivides.html>, 2005. Accessed: 2023-03-06.