

Digit Expansions

Jonas Bayer, Marco David, Abhik Pal and Benedikt Stock

May 26, 2022

Abstract

We formalize how a natural number a can be expanded as

$$a = \sum_{k=0}^l a_k b^k$$

for some base b and prove properties about functions that operate on such expansions. This includes the formalization of concepts such as digit shifts and carries. For a base that is a power of 2 we formalize the binary AND, binary orthogonality and binary masking of two natural numbers. This library on digit expansions builds the basis for the formalization of the DPRM theorem.

Contents

1	Digit functions	2
1.1	Simple properties and equivalences	2
2	Carries in base-b expansions	17
2.1	Definition of carry received at position k	18
2.2	Properties of carries	18
3	Digit-wise Operations	24
3.1	Binary AND	24
3.2	Binary orthogonality	27
3.3	Binary masking	30

1 Digit functions

```
theory Bits-Digits
  imports Main
begin
```

We define the n -th bit of a number in base 2 representation

```
definition nth-bit :: nat  $\Rightarrow$  nat  $\Rightarrow$  nat (infix j 100) where
  nth-bit num k = (num div (2 ^ k)) mod 2
```

as well as the n -th digit of a number in an arbitrary base

```
definition nth-digit :: nat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  nat where
  nth-digit num k base = (num div (base ^ k)) mod base
```

In base 2, the two definitions coincide.

```
lemma nth-digit-base2-equiv: nth-bit a k = nth-digit a k (2::nat)
  by (auto simp add: nth-bit-def nth-digit-def)
```

```
lemma general-digit-base:
```

```
  assumes  $t1 > t2$  and  $b > 1$ 
  shows  $nth-digit (a * b^{t1}) t2 b = 0$ 
```

```
proof -
```

```
  have 1:  $b^{t1} div b^{t2} = b^{(t1-t2)}$  using assms apply auto
    by (metis Suc-lessD less-imp-le less-numeral-extra(3) power-diff)
  have  $b^{t2} dvd b^{t1}$  using  $\langle t1 > t2 \rangle$ 
    by (simp add: le-imp-power-dvd)
```

```
  hence  $(a * b^{t1}) div b^{t2} = a * b^{(t1-t2)}$  using div-mult-swap[of  $b^{t2} b^{t1} a$ ]
  1 by auto
```

```
  thus ?thesis using nth-digit-def assms by auto
```

```
qed
```

```
lemma nth-bit-bounded:  $nth-bit a k \leq 1$ 
  by (auto simp add: nth-bit-def)
```

```
lemma nth-digit-bounded:  $b > 1 \implies nth-digit a k b \leq b-1$ 
  apply (auto simp add: nth-digit-def)
  using less-Suc-eq-le by fastforce
```

```
lemma obtain-smallest:  $P (n::nat) \implies \exists k \leq n. P k \wedge (\forall a < k. \neg(P a))$ 
  by (metis ex-least-nat-le not-less-zero zero-le)
```

1.1 Simple properties and equivalences

Reduce the $nth-digit$ function to (j) if the base is a power of 2

```
lemma digit-gen-pow2-reduct:  $k < c \implies (nth-digit a t (2^c)) j k = a j (c*t+k)$ 
```

```
proof -
```

```
  assume  $k < c$ 
```

```
  then have moddiv:  $(x mod 2^c) j k = x j k$  for  $x$ 
```

proof-
assume *klc*: $k < c$
obtain *a b* **where** *x-def*: $x = a * 2^c + b$ **and** $b < 2^c$
by (*meson mod-div-decomp mod-less-divisor zero-less-numeral zero-less-power*)
then have *bk*: $(x \bmod 2^c) \dot{\vdash} k = b \dot{\vdash} k$ **by** *simp*
have $x \operatorname{div} 2^k = a * 2^{c-k} + b \operatorname{div} 2^k$
using *x-def* **by** (*smt Euclidean-Division.div-eq-0-iff add-diff-inverse-nat add-self-div-2*
div-mult-self3 klc mult.left-commute nat-le-linear not-less power-add
power-eq-0-iff
semiring-normalization-rules(7))
then have $x \dot{\vdash} k = (a * 2^{c-k} + b \operatorname{div} 2^k) \bmod 2$ **by** (*simp add: nth-bit-def*)
then have $x \dot{\vdash} k = b \dot{\vdash} k$ **using** *nth-bit-def klc* **by** (*simp add: mod2-eq-if*)
then show *?thesis* **using** *nth-bit-def bk* **by** *linarith*
qed
have $a \operatorname{div} ((2^c)^t * 2^k) = a \operatorname{div} (2^c)^t \operatorname{div} 2^k$ **using** *div-mult2-eq*
by *blast*
moreover have $a \operatorname{div} (2^c)^t \bmod 2^c \operatorname{div} 2^k \bmod 2 = a \operatorname{div} (2^c)^t \operatorname{div} 2^k \bmod 2$
using *moddiv nth-bit-def* **by** *auto*
ultimately show $(\operatorname{nth-digit} a t (2^c)) \dot{\vdash} k = a \dot{\vdash} (c * t + k)$
using *nth-digit-def nth-bit-def* **by** (*auto simp: power-add power-mult*)
qed

Show equivalence of numbers by equivalence of all their bits (digits)

lemma *aux-even-pow2-factor*: $a > 0 \implies \exists k b. ((a::nat) = (2^k) * b \wedge \operatorname{odd} b)$
proof(*induction a rule: full-nat-induct*)
case (*1 n*)
then show *?case*
proof (*cases odd n*)
case *True*
then show *?thesis* **by** (*metis nat-power-eq-Suc-0-iff power-Suc power-Suc0-right power-commutes*)
next
case *False*
have $(\exists t. n = 2 * t)$ **using** *False* **by** *auto*
then obtain *t* **where** *n-def*: $n = 2 * t$..
then have $t < n$ **using** *1.prem* **by** *linarith*
then have *ih*: $\exists r s. t = 2^r * s \wedge \operatorname{odd} s$ **using** *1 n-def* **by** *simp*
then have $\exists r s. n = 2^{Suc r} * s$ **using** *n-def* **by** *auto*
then show *?thesis* **by** (*metis ih n-def power-commutes semiring-normalization-rules(18)*
semiring-normalization-rules(28))
qed
qed

lemma *aux0-digit-wise-equiv*: $a > 0 \implies (\exists k. \operatorname{nth-bit} a k = 1)$

proof –

assume *a-geq-0*: $a > 0$

consider $(\operatorname{odd}) a \bmod 2 = 1 \mid (\operatorname{even}) a \bmod 2 = 0$ **by** *force*

```

then show ?thesis
proof(cases)
  case odd
  then show ?thesis by (metis div-by-1 nth-bit-def power-0)
next
  case even
  then have bk-def:  $\exists k b. (a = (2^k) * b \wedge \text{odd } b)$ 
    using aux-even-pow2-factor a-geq-0 by simp
  then obtain b k where bk-cond:  $(a = (2^k) * b \wedge \text{odd } b)$  by blast
  then have b = a div (2^k) by simp
  then have digi-b:  $\text{nth-bit } b \ 0 = 1$  using bk-def
    using bk-cond nth-bit-def odd-iff-mod-2-eq-one by fastforce
  then have nth-bit a k = (nth-bit b 0) using nth-bit-def bk-cond by force
  then have nth-bit a k = 1 using digi-b by simp
  then show ?thesis by blast
qed

```

```

lemma aux1-digit-wise-equiv:  $(\forall k. (\text{nth-bit } a \ k = 0)) \iff a = 0$  (is ?P  $\iff$  ?Q)
proof
  assume ?Q
  then show ?P by (simp add: nth-bit-def)
next
  {
    assume a-neq-0:  $\neg ?Q$ 
    then have a > 0 by blast
    then have  $\neg ?P$  using aux0-digit-wise-equiv by (metis zero-neq-one)
  }
  thus ?P  $\implies$  ?Q by blast
qed

```

```

lemma aux2-digit-wise-equiv:  $(\forall r < k. \text{nth-bit } a \ r = 0) \implies (a \bmod 2^k = 0)$ 
proof(induct k)
  case 0
  then show ?case
    by (auto simp add: nth-bit-def)
next
  case (Suc k)
  then show ?case
    by (auto simp add: nth-bit-def)
    (metis dvd-imp-mod-0 dvd-mult-div-cancel dvd-refl even-iff-mod-2-eq-zero
    lessI minus-mod-eq-div-mult minus-mod-eq-mult-div mult-dvd-mono)
qed

```

```

lemma digit-wise-equiv:  $(a = b) \iff (\forall k. \text{nth-bit } a \ k = \text{nth-bit } b \ k)$  (is ?P  $\iff$  ?Q)
proof
  assume ?P
  then show ?Q by simp

```

```

next
{
  assume notP: ¬?P
  have ¬(∀ k. nth-bit a k = nth-bit b k) if ab: a < b for a b
  proof-
    define c::nat where c = b - a
    have b: a+c=b by (auto simp add: c-def ab less-imp-le)
    have ∃ k.(nth-bit c k = 1) using nth-bit-def aux1-digit-wise-equiv
      by (metis c-def not-less0 not-mod-2-eq-1-eq-0 that zero-less-diff)
    then obtain k where k1:nth-bit c k = 1 and k2:∀ r < k. (nth-bit c r ≠ 1)
      by (auto dest: obtain-smallest)
    then have cr0: ∀ r < k. (nth-bit c r = 0) by (simp add: nth-bit-def)
    from aux2-digit-wise-equiv cr0 have c mod 2^k = 0 by auto
    then have a div 2^k mod 2 ≠ b div 2^k mod 2
      by auto (metis b div-plus-div-distrib-dvd-right even-add k1 nth-bit-def
odd-iff-mod-2-eq-one)
    then show ?thesis by (auto simp add: nth-bit-def)
  qed
  from this [of a b] this [of b a] notP have ∀ k. nth-bit a k = nth-bit b k ⇒ a
= b
  using linorder-neqE-nat by auto
}
then show ?Q ==> ?P by auto
qed

```

Represent natural numbers in their binary expansion

```

lemma aux3-digit-sum-repr:
  assumes b < 2^r
  shows (a*2^r + b) j r = (a*2^r) j r
  by (auto simp add: nth-bit-def assms)

```

```

lemma aux2-digit-sum-repr:
  assumes n < 2^c r < c
  shows (a*2^c+n) j r = n j r
  proof -
    have [simp]: ⟨a*(2::nat) ^ c div 2 ^ r = a*2 ^ (c - r)⟩
      using assms(2)
    by (auto simp: less-iff-Suc-add
monoid-mult-class.power-add ac-simps)
  show ?thesis
    using assms
  by (auto simp add: nth-bit-def
div-plus-div-distrib-dvd-left
le-imp-power-dvd
simp flip: mod-add-left-eq)
  qed

```

```

lemma aux1-digit-sum-repr:
  assumes n < 2^c r < c

```

shows $(\sum k < c. ((n \text{ ; } k) * 2^{\wedge} k)) \text{ ; } r = n \text{ ; } r$
proof –
define a **where** $a \equiv (\sum k = 0..< \text{Suc } r). ((n \text{ ; } k) * 2^{\wedge} k)$
define d **where** $d \equiv (\sum k = \text{Suc } r..< c. ((n \text{ ; } k) * 2^{\wedge} k))$
define e **where** $e \equiv (\sum k = \text{Suc } r..< c. ((n \text{ ; } k) * 2^{\wedge} (k - \text{Suc } r)))$
define b **where** $b \equiv (\sum k = 0..< r. ((n \text{ ; } k) * 2^{\wedge} k))$
have ad : $(\sum k = 0..< c. ((n \text{ ; } k) * 2^{\wedge} k)) = a + d$
using $a\text{-def } d\text{-def } \text{assms}$
by ($\text{metis } (\text{no-types, lifting}) \text{Suc-leI } a\text{-def } \text{assms}(2) \text{d-def } \text{sum.atLeastLessThan-concat}$
 zero-le)
have $d = (\sum k = \text{Suc } r..< c. (2^{\wedge} (\text{Suc } r) * (n \text{ ; } k * 2^{\wedge} (k - \text{Suc } r))))$
using $d\text{-def } \text{apply } (\text{simp})$
apply ($\text{rule } \text{sum.cong; auto } \text{simp: algebra-simps}$)
by ($\text{metis } \text{add-Suc } \text{le-add-diff-inverse } \text{numerals}(2) \text{power-Suc } \text{power-add}$)
hence $d2r$: $d = 2^{\wedge} \text{Suc } r * e$ **using** $d\text{-def } e\text{-def}$
 $\text{sum-distrib-left[of } 2^{\wedge} (\text{Suc } r) \lambda k. (n \text{ ; } k) * 2^{\wedge} (k - \text{Suc } r) \{\text{Suc } r..< c\}]$ **by** auto
have $(\sum k < c. ((n \text{ ; } k) * 2^{\wedge} k)) = a + 2^{\wedge} \text{Suc } r * e$ **by** ($\text{simp } \text{add: } d2r \text{ad } \text{lessThan-atLeast0}$)
moreover **have** $(\sum k = 0..< \text{Suc } r. ((n \text{ ; } k) * 2^{\wedge} k)) < 2^{\wedge} \text{Suc } r$ **using** assms
proof($\text{induct } r$)
case 0
then show $?case$
proof –
have $n \text{ ; } 0 < \text{Suc } 1$
by ($\text{metis } (\text{no-types}) \text{atLeastLessThan-empty } \text{atLeastLessThan-iff } \text{lessI}$
 linorder-not-less
 nth-bit-bounded)
then show $?thesis$
by simp
qed
next
case $(\text{Suc } r)$
have $r2$: $n \text{ ; } \text{Suc } r * 2^{\wedge} \text{Suc } r \leq 2^{\wedge} \text{Suc } r$ **using** $\text{nth-bit-bounded } \text{by}$
 simp
have $(\sum k = 0..< \text{Suc } r. ((n \text{ ; } k) * 2^{\wedge} k)) < 2^{\wedge} \text{Suc } r$
using $\text{Suc.hyps } \text{using } \text{Suc.prems}(2) \text{Suc-lessD } \text{assms}(1)$ **by** blast
then show $?case$ **using** $r2$
by ($\text{smt } \text{Suc-leI } \text{Suc-le-lessD } \text{add-Suc } \text{add-le-mono } \text{mult-2 } \text{power-Suc}$
 $\text{sum.atLeast0-lessThan-Suc}$)
qed
then have $a < 2^{\wedge} \text{Suc } r$ **using** $a\text{-def } \text{by } \text{blast}$
ultimately have ar : $(a + d) \text{ ; } r = a \text{ ; } r$ **using** $d2r$
by ($\text{metis } (\text{no-types, lifting}) \text{aux2-digit-sum-repr } \text{lessI } \text{semiring-normalization-rules}(24)$
 $\text{semiring-normalization-rules}(7)$)
have ab : $a = (n \text{ ; } r) * 2^{\wedge} r + b$ **using** $a\text{-def } b\text{-def } d\text{-def } \text{by } \text{simp}$
have $(\sum k = 0..< r. ((n \text{ ; } k) * 2^{\wedge} k)) < 2^{\wedge} r$
proof($\text{induct } r$)
case 0

```

    then show ?case by auto
  next
    case (Suc r)
    have r2:  $n \dot{\vdash} r * 2^{\wedge} r \leq 2^{\wedge} r$  using nth-bit-bounded by simp
    have ( $\sum k = 0..<r. n \dot{\vdash} k * 2^{\wedge} k$ ) <  $2^{\wedge} r$  using Suc.hyps by auto
    then show ?case apply simp using r2 by linarith
  qed
  then have b:  $b < 2^{\wedge} r$  using b-def by blast
  then have a  $\dot{\vdash}$  r =  $((n \dot{\vdash} r) * 2^{\wedge} r) \dot{\vdash} r$  using ab aux3-digit-sum-repr by simp
  then have a  $\dot{\vdash}$  r =  $(n \dot{\vdash} r)$  using nth-bit-def by simp
  then show ?thesis using ar a-def by (simp add: ad lessThan-atLeast0)
qed

lemma digit-sum-repr:
  assumes  $n < 2^{\wedge} c$ 
  shows  $n = (\sum k < c. ((n \dot{\vdash} k) * 2^{\wedge} k))$ 
proof -
  have  $\forall k. (c \leq k \longrightarrow n < 2^{\wedge} k)$  using assms less-le-trans by fastforce
  then have nik:  $\forall k. (k \geq c \longrightarrow (n \dot{\vdash} k = 0))$  by (auto simp add: nth-bit-def)
  have ( $\sum r < c. ((n \dot{\vdash} r) * (2::nat)^{\wedge} r)$ ) <  $(2::nat)^{\wedge} c$ 
  proof (induct c)
    case 0
    then show ?case by simp
  next
    case (Suc c)
    then show ?case
      using nth-bit-bounded
      add-mono-thms-linordered-field[of  $(n \dot{\vdash} c) * 2^{\wedge} c$   $2^{\wedge} c$   $(\sum r < c. n \dot{\vdash} r * 2^{\wedge} r)$   $2^{\wedge} c$ ]
      by simp
  qed
  then have  $\forall k. k \geq c \longrightarrow (\sum r < c. ((n \dot{\vdash} r) * 2^{\wedge} r)) \dot{\vdash} k = 0$ 
    using less-le-trans by (auto simp add: nth-bit-def) fastforce
  then have  $\forall r \geq c. (n \dot{\vdash} r = (\sum k < c. ((n \dot{\vdash} k) * 2^{\wedge} k)) \dot{\vdash} r)$  by (simp add: nik)
  moreover have  $\forall r < c. (n \dot{\vdash} r = (\sum k < c. ((n \dot{\vdash} k) * 2^{\wedge} k)) \dot{\vdash} r)$  using aux1-digit-sum-repr
  assms by simp
  ultimately have  $\forall r. (\sum k < c. ((nth-bit n k) * 2^{\wedge} k)) \dot{\vdash} r = n \dot{\vdash} r$  by (metis not-less)
  then show ?thesis using digit-wise-equiv by presburger
qed

lemma digit-sum-repr-variant:
   $n = (\sum k < n. ((nth-bit n k) * 2^{\wedge} k))$ 
  using less-exp digit-sum-repr by auto

lemma digit-sum-index-variant:
   $r > n \longrightarrow ((\sum k < n. ((n \dot{\vdash} k) * 2^{\wedge} k)) = (\sum k < r. (n \dot{\vdash} k) * 2^{\wedge} k))$ 
proof -
  have  $\forall r. (2^{\wedge} r > r)$  using less-exp by simp
  then have pow2:  $\forall r. (r > n \longrightarrow 2^{\wedge} r > n)$  using less-trans by blast

```

then have $\forall r. (r > n \longrightarrow (n \text{ j } r = 0))$ **by** (*auto simp add: nth-bit-def*)
then have $\forall r. r > n \longrightarrow (n \text{ j } r) * 2^{\hat{r}} = 0$ **by** *auto*
then show *?thesis* **using** *digit-sum-repr digit-sum-repr-variant pow2* **by** *auto*
qed

Digits are preserved under shifts

lemma *digit-shift-preserves-digits*:
assumes $b > 1$
shows $\text{nth-digit } (b * y) \text{ (Suc } t) b = \text{nth-digit } y \text{ } t b$
using *nth-digit-def assms* **by** *auto*

lemma *digit-shift-inserts-zero-least-siginificant-digit*:
assumes $t > 0$ **and** $b > 1$
shows $\text{nth-digit } (1 + b * y) \text{ } t b = \text{nth-digit } (b * y) \text{ } t b$
using *nth-digit-def assms* **apply** *auto*

proof –

assume $0 < t$
assume $\text{Suc } 0 < b$
hence $\text{Suc } (b * y) \text{ mod } b = 1$
by (*simp add: Suc-times-mod-eq*)
hence $b * y \text{ div } b = \text{Suc } (b * y) \text{ div } b$
using $\langle b > 1 \rangle$ **by** (*metis (no-types) One-nat-def diff-Suc-Suc gr-implies-not0 minus-mod-eq-div-mult mod-mult-self1-is-0 nonzero-mult-div-cancel-right*)
then show $\text{Suc } (b * y) \text{ div } b^{\hat{t}} \text{ mod } b = b * y \text{ div } b^{\hat{t}} \text{ mod } b$
using $\langle t > 0 \rangle$ **by** (*metis Suc-pred div-mult2-eq power-Suc*)

qed

Represent natural numbers in their base-b digitwise expansion

lemma *aux3-digit-gen-sum-repr*:
assumes $d < b^{\hat{r}}$ **and** $b > 1$
shows $\text{nth-digit } (a * b^{\hat{r}} + d) \text{ } r b = \text{nth-digit } (a * b^{\hat{r}}) \text{ } r b$
using $\langle b > 1 \rangle$ **by** (*auto simp: nth-digit-def assms*)

lemma *aux2-digit-gen-sum-repr*:
assumes $n < b^{\hat{c}}$ $r < c$
shows $\text{nth-digit } (a * b^{\hat{c} + n}) \text{ } r b = \text{nth-digit } n \text{ } r b$

proof –

have [*simp*]: $\langle a * b^{\hat{c}} \text{ div } b^{\hat{r}} = a * b^{\hat{c} - r} \rangle$
using *assms(2)*
by (*auto simp: less-iff-Suc-add monoid-mult-class.power-add ac-simps*)

show *?thesis*

using *assms*
by (*auto simp add: nth-digit-def div-plus-div-distrib-dvd-left le-imp-power-dvd simp flip: mod-add-left-eq*)

qed

lemma *aux1-digit-gen-sum-repr*:
assumes $n < b^{\wedge}c$ $r < c$ **and** $b > 1$
shows $\text{nth-digit } (\sum k < c. ((\text{nth-digit } n \ k \ b) * b^{\wedge}k)) \ r \ b = \text{nth-digit } n \ r \ b$
proof –
define a **where** $a \equiv (\sum k = 0.. < \text{Suc}(r). ((\text{nth-digit } n \ k \ b) * b^{\wedge}k))$
define d **where** $d \equiv (\sum k = \text{Suc}(r).. < c. ((\text{nth-digit } n \ k \ b) * b^{\wedge}k))$
define e **where** $e \equiv (\sum k = \text{Suc } r.. < c. ((\text{nth-digit } n \ k \ b) * b^{\wedge}(k - \text{Suc}(r))))$
define f **where** $f \equiv (\sum k = 0.. < r. ((\text{nth-digit } n \ k \ b) * b^{\wedge}k))$
have ad : $(\sum k = 0.. < c. ((\text{nth-digit } n \ k \ b) * b^{\wedge}k)) = a + d$
using *a-def d-def assms*
by (*metis (no-types, lifting) Suc-leI a-def assms(2) d-def sum.atLeastLessThan-concat zero-le*)
have $d = (\sum k = \text{Suc}(r).. < c. (b^{\wedge}(\text{Suc } r) * (\text{nth-digit } n \ k \ b * b^{\wedge}(k - \text{Suc } r))))$
using *d-def apply (auto) apply (rule sum.cong; auto simp: algebra-simps)*
by (*metis add-Suc le-add-diff-inverse power-Suc power-add*)
hence $d2r$: $d = b^{\wedge}\text{Suc}(r) * e$ **using** *d-def e-def sum-distrib-left* [of $b^{\wedge}(\text{Suc } r)$
 $\lambda k. (\text{nth-digit } n \ k \ b) * b^{\wedge}(k - \text{Suc } r) \{ \text{Suc } r.. < c \}$] **by** *auto*
have $(\sum k < c. ((\text{nth-digit } n \ k \ b) * b^{\wedge}k)) = a + b^{\wedge}\text{Suc}(r) * e$
by (*simp add: d2r ad lessThan-atLeast0*)
moreover **have** $(\sum k = 0.. < \text{Suc}(r). ((\text{nth-digit } n \ k \ b) * b^{\wedge}k)) < b^{\wedge} \text{Suc}(r)$ **using**
assms
proof(*induct r*)
case 0
then show *?case*
proof –
have $\text{nth-digit } n \ 0 \ b < b$
using *nth-digit-bounded*[of $b \ n \ 0$] $\langle b > 1 \rangle$ **by** *auto*
then show *?thesis*
by *simp*
qed
next
case $(\text{Suc } r)$
have $r2$: $(\text{nth-digit } n \ (\text{Suc } r) \ b) * b^{\wedge} \text{Suc}(r) \leq (b - 1) * b^{\wedge} \text{Suc}(r)$
using *nth-digit-bounded*[of $b \ n \ \text{Suc } r$] $\langle b > 1 \rangle$ **by** *auto*
moreover **have** $(\sum k = 0.. < \text{Suc}(r). ((\text{nth-digit } n \ k \ b) * b^{\wedge}k)) < b^{\wedge} \text{Suc}(r)$
using *Suc.hyps using Suc.prem(2) Suc-lessD assms(1) \langle b > 1 \rangle* **by** *blast*
ultimately **have** $(\text{nth-digit } n \ (\text{Suc } r) \ b) * b^{\wedge} \text{Suc}(r)$
 $+ (\sum k = 0.. < \text{Suc}(r). ((\text{nth-digit } n \ k \ b) * b^{\wedge}k)) < b^{\wedge} \text{Suc} (\text{Suc } r)$
using *assms(3) mult-eq-if* **by** *auto*
then show *?case* **by** *auto*
qed
hence $a < b^{\wedge}\text{Suc}(r)$ **using** *a-def* **by** *blast*
ultimately **have** ar : $\text{nth-digit } (a + d) \ r \ b = \text{nth-digit } a \ r \ b$ **using** $d2r$
by (*metis (no-types, lifting) aux2-digit-gen-sum-repr lessI semiring-normalization-rules(24)*
 $\text{semiring-normalization-rules}(7)$)
have ab : $a = (\text{nth-digit } n \ r \ b) * b^{\wedge}r + f$ **using** *a-def f-def d-def* **by** *simp*

```

have ( $\sum_{k=0..<r} ((nth-digit\ n\ k\ b) * b^k) < b^r$ )
proof(induct r)
  case 0
  then show ?case by auto
next
case (Suc r)
have r2:  $nth-digit\ n\ r\ b * b^r \leq (b-1) * b^r$ 
  using nth-digit-bounded[of b] <b>1> by auto
have ( $\sum_{k=0..<r} nth-digit\ n\ k\ b * b^k < b^r$ ) using Suc.hyps by auto
then show ?case using r2 assms(3) mult-eq-if by auto
qed
hence f:  $f < b^r$  using f-def by blast
hence  $nth-digit\ a\ r\ b = (nth-digit\ (nth-digit\ n\ r\ b * b^r)\ r\ b)$ 
  using ab aux3-digit-gen-sum-repr <b>1> by simp
hence  $nth-digit\ a\ r\ b = nth-digit\ n\ r\ b$ 
  using nth-digit-def <b>1> by simp
then show ?thesis using ar a-def by (simp add: ad lessThan-atLeast0)
qed

lemma aux-gen-b-factor:  $a > 0 \implies b > 1 \implies \exists k\ c. ((a::nat) = (b^k) * c \wedge \neg(c \text{ mod } b = 0))$ 
proof(induction a rule: full-nat-induct)
  case (1 n)
  show ?case
  proof(cases n mod b = 0)
    case True
    then obtain t where n-def:  $n = b * t$  by blast
    hence  $t < n$ 
      using 1 by auto
    with 1 have ih: $\exists r\ s. t = b^r * s \wedge \neg(s \text{ mod } b = 0)$ 
      by (metis Suc-leI grOI mult-0-right n-def)
    hence  $\exists r\ s. n = b^{Suc\ r} * s$  using n-def by auto
    then show ?thesis by (metis ih mult.commute mult.left-commute n-def power-Suc)
  next
  case False
  then show ?thesis by (metis mult.commute power-0 power-Suc power-Suc0-right)
  qed
qed

lemma aux0-digit-wise-gen-equiv:
  assumes  $b > 1$  and a-geq-0:  $a > 0$ 
  shows ( $\exists k. nth-digit\ a\ k\ b \neq 0$ )
proof(cases a mod b = 0)
  case True
  hence  $\exists k\ c. a = (b^k) * c \wedge \neg(c \text{ mod } b = 0)$ 
    using aux-gen-b-factor a-geq-0 assms by simp
  then obtain c k where ck-cond: $a = (b^k) * c \wedge \neg(c \text{ mod } b = 0)$  by blast
  hence c-cond: $c = a \text{ div } (b^k)$  using a-geq-0 by auto
  hence digi-b: $nth-digit\ c\ 0\ b \neq 0$ 

```

```

    using ck-cond nth-digit-def by force
  hence nth-digit a k b = nth-digit c 0 b
    using nth-digit-def c-cond by simp
  hence nth-digit a k b ≠ 0 using digi-b by simp
  then show ?thesis by blast next
  case False
  then show ?thesis
    by (metis div-by-1 nth-digit-def power.simps(1))
qed

```

```

lemma aux1-digit-wise-gen-equiv:
  assumes b > 1
  shows (∀ k. (nth-digit a k b = 0)) ↔ a = 0 (is ?P ↔ ?Q)
proof
  assume ?Q
  then show ?P by (simp add: nth-digit-def)
next
  {
    assume a-neq-0: ¬?Q
    hence a > 0 by blast
    hence ¬?P using aux0-digit-wise-gen-equiv ⟨b > 1⟩ by auto
  } from this show ?P ⇒ ?Q by blast
qed

```

```

lemma aux2-digit-wise-gen-equiv: (∀ r < k. nth-digit a r b = 0) → (a mod b^k = 0)
proof(induct k)
  case 0
  then show ?case by auto
next
  case (Suc k)
  then show ?case apply(auto simp add: nth-digit-def)
    using dvd-imp-mod-0 dvd-mult-div-cancel dvd-refl
      lessI minus-mod-eq-div-mult minus-mod-eq-mult-div mult-dvd-mono
    by (metis mod-0-imp-dvd)
qed

```

Two numbers are the same if and only if their digits are the same

```

lemma digit-wise-gen-equiv:
  assumes b > 1
  shows (x = y) ↔ (∀ k. nth-digit x k b = nth-digit y k b) (is ?P ↔ ?Q)
proof
  assume ?P
  then show ?Q by simp
next{
  assume notP: ¬?P
  have ¬(∀ k. nth-digit x k b = nth-digit y k b) if xy: x < y for x y
  proof-
    define c::nat where c = y - x

```

```

have  $y: x+c=y$  by (auto simp add: c-def xy less-imp-le)
have  $\exists k.(nth\_digit\ c\ k\ b \neq 0)$ 
  using nth-digit-def  $\langle b>1 \rangle$  aux0-digit-wise-gen-equiv
  by (metis c-def that zero-less-diff)
then obtain  $k$  where  $k1:nth\_digit\ c\ k\ b \neq 0$ 
  and  $k2:\forall r < k. (nth\_digit\ c\ r\ b = 0)$ 
  apply(auto dest: obtain-smallest) done
hence  $cr0: \forall r < k. (nth\_digit\ c\ r\ b = 0)$  by (simp add: nth-digit-def)
from aux2-digit-wise-gen-equiv  $cr0$  have  $c \bmod b^k = 0$  by auto
hence  $x \bmod b^k \neq y \bmod b^k$ 
  using  $y\ k1\ \langle b>1 \rangle$  aux1-digit-wise-gen-equiv[ $of\ b\ c$ ] nth-digit-def apply auto
proof -
  fix  $ka :: nat$ 
  assume  $a1: b^k \mid c$ 
  assume  $a2: x \bmod b^k = (x + c) \bmod b^k$ 
  assume  $a3: y = x + c$ 
  assume  $a4: \bigwedge num\ k\ base. nth\_digit\ num\ k\ base$ 
     $= num \bmod b^k$ 
have  $f5: \forall n\ na. (na::nat) + n - na = n$ 
  by simp
have  $f6: x \bmod b^k + c \bmod b^k = y \bmod b^k$ 
using  $a3\ a1$  by (simp add: add.commute mod-plus-mod-distrib-dvd-left)
have  $f7: \forall n. (x \bmod b^k + n) \bmod b = (y \bmod b^k + n) \bmod b$ 
  using  $a3\ a2$  by (metis add.commute mod-add-right-eq)
have  $\forall n\ na\ nb. ((nb::nat) \bmod na + n - (nb + n) \bmod na) \bmod$ 
 $na = 0$ 
  by (metis (no-types) add.commute minus-mod-eq-mult-div
    mod-add-right-eq
    mod-mult-self1-is-0)
then show  $c \bmod b^k = 0$ 
using  $f7\ f6\ f5\ a4$  by (metis (no-types)  $k1$ )
qed
then show ?thesis by (auto simp add: nth-digit-def)
qed
from this [of  $x\ y$ ] this [of  $y\ x$ ] notP
have  $\forall k. nth\_digit\ x\ k\ b = nth\_digit\ y\ k\ b \implies x = y$  apply(auto)
using linorder-neqE-nat by blast} then show ?Q  $\implies$  ?P by auto
qed

```

A number is equal to the sum of its digits multiplied by powers of two

lemma digit-gen-sum-repr:

assumes $n < b^c$ **and** $b > 1$

shows $n = (\sum_{k < c} (nth_digit\ n\ k\ b) * b^k)$

proof -

have $1: (c \leq k \implies nth_digit\ n\ k\ b = 0)$ **for** k **using** $assms$ less-le-trans **by** fastforce

hence $nik: (c \leq k \implies nth_digit\ n\ k\ b = 0)$ **for** k

by (auto simp add: nth-digit-def)

have $(\sum_{r < c} (nth_digit\ n\ r\ b) * b^r) < b^c$ **apply**(induct c , auto)

subgoal **for** c

proof –
assume $IH: (\sum r < c. nth_digit\ n\ r\ b * b^{\wedge} r) < b^{\wedge} c$
have $bound: (nth_digit\ n\ c\ b) * b^{\wedge} c \leq (b-1) * b^{\wedge} c$
using $nth_digit_bounded\ \langle b > 1 \rangle$ **by** $auto$
thus $?thesis$ **using** $assms\ IH$
by $(metis\ (no_types,\ lifting)\ bound\ add_mono_thms_linordered_field(1)\ add_mono_thms_linordered_field(5)\ le_less\ mult_eq_if\ not_one_le_zero)$
qed
done
hence $k \geq c \longrightarrow nth_digit\ (\sum r < c. ((nth_digit\ n\ r\ b) * b^{\wedge} r))\ k\ b = 0$ **for** k
apply $(auto\ simp\ add: nth_digit_def)$ **using** $less_le_trans\ assms(2)$ **by** $fastforce$
hence $\forall r \geq c. (nth_digit\ n\ r\ b$
 $= nth_digit\ (\sum k < c. ((nth_digit\ n\ k\ b) * b^{\wedge} k))\ r\ b)$ **by** $(simp\ add: nik)$
moreover **have** $\forall r < c. (nth_digit\ n\ r\ b$
 $= nth_digit\ (\sum k < c. ((nth_digit\ n\ k\ b) * b^{\wedge} k))\ r\ b)$
using $aux1_digit_gen_sum_repr\ assms$ **by** $simp$
ultimately **have** $\forall r. nth_digit\ (\sum k < c. ((nth_digit\ n\ k\ b) * b^{\wedge} k))\ r\ b$
 $= nth_digit\ n\ r\ b$ **by** $(metis\ not_less)$
then **show** $?thesis$
using $digit_wise_gen_equiv[of\ b\ (\sum k < c. nth_digit\ n\ k\ b * b^{\wedge} k)\ n]\ \langle b > 1 \rangle$ **by**
 $auto$
qed

lemma $digit_gen_sum_repr_variant:$

assumes $b > 1$
shows $n = (\sum k < n. ((nth_digit\ n\ k\ b) * b^{\wedge} k))$
proof –
have $n < b^{\wedge} n$ **using** $\langle b > 1 \rangle$ **apply** $(induct\ n,\ auto)$ **by** $(simp\ add: less_trans_Suc)$

then **show** $?thesis$ **using** $digit_gen_sum_repr\ \langle b > 1 \rangle$ **by** $auto$
qed

lemma $digit_gen_sum_index_variant:$

assumes $b > 1$ **shows** $r > n \implies$
 $(\sum k < n. ((nth_digit\ n\ k\ b) * b^{\wedge} k)) = (\sum k < r. (nth_digit\ n\ k\ b) * b^{\wedge} k)$
proof –
assume $r > n$
have $b^{\wedge} r > r$ **for** r **using** $\langle b > 1 \rangle$ **by** $(induction\ r,\ auto\ simp\ add: less_trans_Suc)$
hence $powb: \forall r. (r > n \longrightarrow b^{\wedge} r > n)$ **using** $less_trans$ **by** $auto$
hence $r > n \longrightarrow (nth_digit\ n\ r\ b = 0)$ **for** r
by $(auto\ simp\ add: nth_digit_def)$
hence $r > n \longrightarrow (nth_digit\ n\ r\ b) * b^{\wedge} r = 0$ **for** r **by** $auto$
then **show** $?thesis$ **using** $digit_gen_sum_repr\ digit_gen_sum_repr_variant\ powb\ \langle n < r \rangle$ $assms$ **by** $auto$
qed

nth_digit extracts coefficients from a base-b digitwise expansion

lemma $nth_digit_gen_power_series:$

fixes $c\ b\ k\ q$

```

defines  $b \equiv 2^{\wedge}(Suc\ c)$ 
assumes  $bound: \forall k. (f\ k) < b$ 
shows  $nth\_digit\ (\sum_{k=0..q}. (f\ k) * b^{\wedge}k)\ t\ b = (if\ t \leq q\ then\ (f\ t)\ else\ 0)$ 
proof (induction q arbitrary: t)
  case 0
  have  $b > 1$  using  $b\_def$ 
    using  $one\_less\_numeral\_iff\ power\_gt1\ semiring\_norm(76)$  by  $blast$ 
  have  $f\ 0 < b$  using  $bound$  by  $auto$ 
  hence  $t > 0 \longrightarrow f\ 0 < b^{\wedge}t$  using  $\langle b > 1 \rangle$ 
    using  $bound\ less\_imp\_le\_nat\ less\_le\_trans$  by ( $metis\ self\_le\_power$ )
  thus  $?case$  using  $nth\_digit\_def\ bound$  by  $auto$ 
next
  case ( $Suc\ q$ )
  thus  $?case$ 
  proof ( $cases\ t \leq Suc\ q$ )
    case  $True$ 
    have  $f\_le\_bound: f\ k \leq b - 1$  for  $k$  using  $bound$  apply  $auto$ 
    by ( $metis\ Suc\_pred\ b\_def\ less\_Suc\_eq\_le\ numeral\_2\_eq\_2\ zero\_less\_Suc\ zero\_less\_power$ )
    have  $series\_bound: (\sum_{k=0..q}. f\ k * b^{\wedge}k) < b^{\wedge}(Suc\ q)$ 
    apply ( $induct\ q$ )
    subgoal using  $bound$  by ( $simp\ add: less\_imp\_le\_nat$ )
    subgoal for  $q$ 
    proof -
      assume  $asm: (\sum_{k=0..q}. f\ k * b^{\wedge}k) < b^{\wedge}Suc\ q$ 
      have  $(\sum_{k=0..q}. f\ k * b^{\wedge}k) + f\ (Suc\ q) * (b * b^{\wedge}q)$ 
         $\leq (\sum_{k=0..q}. f\ k * b^{\wedge}k) + (b - 1) * (b * b^{\wedge}q)$  using  $f\_le\_bound$ 
by  $auto$ 
      also have  $\dots < b^{\wedge}(Suc\ q) + (b - 1) * (b * b^{\wedge}q)$  using  $asm$  by  $auto$ 
      also have  $\dots \leq b * b * b^{\wedge}q$  apply  $auto$ 
      by ( $metis\ One\_nat\_def\ Suc\_neq\_Zero\ b\_def\ eq\_imp\_le\ mult.assoc\ mult\_eq\_iff$ 
         $numerals(2)$ 
         $power\_not\_zero$ )
      finally show  $?thesis$  using  $asm$  by  $auto$ 
    qed
  done
  hence  $nth\_digit\ ((\sum_{k=0..q}. f\ k * b^{\wedge}k) + f\ (Suc\ q) * (b * b^{\wedge}q))\ t\ b = f\ t$ 
  using  $Suc\ nth\_digit\_def$  apply ( $cases\ t = Suc\ q, auto$ )
  subgoal using  $Suc\_n\_not\_le\_n\ add.commute\ add.left\_neutral\ b\_def\ bound$ 
   $div\_mult\_self1$ 
   $less\_imp\_le\_nat\ less\_mult\_imp\_div\_less\ mod\_less\ not\_one\_le\_zero\ one\_less\_numeral\_iff$ 
   $one\_less\_power\ power\_Suc\ semiring\_norm(76)\ zero\_less\_Suc$  by  $auto$ 
subgoal
proof -
  assume  $t \neq Suc\ q$ 
  hence  $t < Suc\ q$  using  $True$  by  $auto$ 
  hence  $nth\_digit\ (f\ (Suc\ q) * b^{\wedge}Suc\ q + (\sum_{k=0..q}. f\ k * b^{\wedge}k))\ t\ b$ 
     $= nth\_digit\ (\sum_{k=0..q}. f\ k * b^{\wedge}k)\ t\ b$ 
    using  $aux2\_digit\_gen\_sum\_repr[of\ \sum_{k=0..q}. f\ k * b^{\wedge}k\ b\ Suc\ q\ t$ 

```

$f (Suc\ q)]$ series-bound by auto
hence $((\sum k = 0..q. f\ k * b^{\wedge}k) + f (Suc\ q) * (b * b^{\wedge}q)) \text{ div } b^{\wedge}t \text{ mod } b$
 $= (\sum k = 0..q. f\ k * b^{\wedge}k) \text{ div } b^{\wedge}t \text{ mod } b$
using *nth-digit-def* by (*auto simp:add.commute*)
thus *?thesis* **using** *Suc[of t] nth-digit-def True <t ≠ Suc q>* by auto
qed
done
thus *?thesis* **using** *True* by auto
next
case *False*
hence $t \geq Suc (Suc\ q)$ by auto
have *f-le-bound*: $f\ k \leq b-1$ for k **using** *bound* **apply** auto
by (*metis Suc-pred b-def less-Suc-eq-le numeral-2-eq-2 zero-less-Suc zero-less-power*)
have *bound*: $(\sum k = 0..q. f\ k * b^{\wedge}k) < b^{\wedge}(Suc\ q)$ for q
apply (*induct q*)
subgoal **using** *bound* by (*simp add: less-imp-le-nat*)
subgoal for q
proof –
assume *asm*: $(\sum k = 0..q. f\ k * b^{\wedge}k) < b^{\wedge}Suc\ q$
have $(\sum k = 0..q. f\ k * b^{\wedge}k) + f (Suc\ q) * (b * b^{\wedge}q)$
 $\leq (\sum k = 0..q. f\ k * b^{\wedge}k) + (b-1) * (b * b^{\wedge}q)$ **using** *f-le-bound*
by auto
also **have** $\dots < b^{\wedge}(Suc\ q) + (b-1) * (b * b^{\wedge}q)$ **using** *asm* by auto
also **have** $\dots \leq b * b * b^{\wedge}q$ **apply** auto
by (*metis One-nat-def Suc-neq-Zero b-def eq-imp-le mult.assoc mult-eq-if numerals(2) power-not-zero*)
finally **show** *?thesis* **using** *asm* by auto
qed
done
have $(\sum k = 0..q. f\ k * b^{\wedge}k) + f (Suc\ q) * (b * b^{\wedge}q) < (b^{\wedge}Suc (Suc\ q))$
using *bound[of Suc q]* by auto
also **have** $\dots \leq b^{\wedge}t$ **using** $\langle t \geq Suc (Suc\ q) \rangle$
apply auto by (*metis b-def nat-power-less-imp-less not-le numeral-2-eq-2 power-Suc zero-less-Suc zero-less-power*)
finally **show** *?thesis* **using** $\langle t \geq Suc (Suc\ q) \rangle$ *bound[of Suc q] nth-digit-def* by auto
qed
qed

Equivalence condition for the *nth-digit* function [1] (see equation 2.29)

lemma *digit-gen-equiv*:

assumes $b > 1$
shows $d = \text{nth-digit } a\ k\ b \iff (\exists x. \exists y. (a = x * b^{\wedge}(k+1) + d * b^{\wedge}k + y \wedge d < b \wedge y < b^{\wedge}k))$
(is *?P* \iff *?Q*)
proof

```

assume p: ?P
then show ?Q
proof(cases k<a)
  case True

  have ( $\sum i<k.((nth-digit a i b)*b^i)$ ) < b^k
  proof(induct k)
    case 0
    then show ?case by auto
  next
  case (Suc k)
  have ( $\sum i<Suc k. nth-digit a i b * b^i$ ) = (nth-digit a k b) * b^k
    + ( $\sum i<k.((nth-digit a i b)*b^i)$ ) by simp
  moreover have (nth-digit a k b) * b^k ≤ (b-1) * b^k
    using assms mult-le-mono nth-digit-bounded by auto
  moreover have ( $\sum i<k.((nth-digit a i b)*b^i)$ ) < b^(Suc k)
    using Suc.hyps assms order.strict-trans2 by fastforce
  ultimately show ?case using assms using Suc.hyps mult-eq-if by auto
qed
moreover define y where y = ( $\sum i<k.((nth-digit a i b)*b^i)$ )
ultimately have 3: y < b^k by blast

```

have 2: $d < b$ using nth-digit-bounded[of b a k] p assms by linarith

```

define x where x = ( $\sum i=Suc k..<a. ((nth-digit a i b)*b^{i-Suc k})$ )
have a = ( $\sum i<a.((nth-digit a i b)*b^i)$ ) using assms digit-gen-sum-repr-variant
by blast
hence s:a = y + d * b^k
  + ( $\sum i=Suc k..<a.((nth-digit a i b)*b^i)$ ) using True y-def p
  by (metis (no-types, lifting) Suc-leI atLeast0LessThan gr-implies-not0
    linorder-not-less sum.atLeastLessThan-concat sum.lessThan-Suc)
have ( $\sum n = Suc k..<a. nth-digit a n b * b^{(n - Suc k) * b^{Suc k}}$ )
  = ( $\sum i = Suc k..<a. nth-digit a i b * b^i$ )
  apply (rule sum.cong; auto simp: algebra-simps)
by (metis Suc-le-lessD Zero-not-Suc add-diff-cancel-left' diff-Suc-1 diff-Suc-Suc
  less-imp-Suc-add power-add power-eq-if)
hence x * b^(k+1) = ( $\sum i=Suc k..<a.((nth-digit a i b)*b^i)$ )
  using x-def sum-distrib-right[of λi. (nth-digit a i b) * b^(i - Suc k)] by simp
hence a = x * b^(k+1) + d*b^k + y using s by auto
thus ?thesis using 2 3 by blast
next
case False
then have a < b^k using assms power-gt-expt[of b k] by auto
moreover have d = 0 by (simp add: calculation nth-digit-def p)
ultimately show ?thesis

```



```

    using assms by force
  qed
next
  assume ?Q
  then obtain x y where conds:  $a = x * b^{(k+1)} + d * b^k + y \wedge d < b \wedge y < b^k$  by auto
  hence  $nth\_digit\ a\ k\ b = nth\_digit(x * b^{(k+1)} + d * b^k)\ k\ b$ 
    using aux3-digit-gen-sum-repr[of y b k x*b + d] assms by (auto simp: algebra-simps)
  hence  $nth\_digit\ a\ k\ b = nth\_digit(d * b^k)\ k\ b$ 
    using aux2-digit-gen-sum-repr[of  $d * b^k\ b\ k+1\ k\ x$ ] conds by auto
  then show ?P using conds nth-digit-def by simp
qed

end
theory Carries
  imports Bits-Digits
begin

```

2 Carries in base-b expansions

Some auxiliary lemmas

```

lemma rev-induct[consumes 1, case-names base step]:
  fixes i k :: nat
  assumes le:  $i \leq k$ 
    and base:  $P\ k$ 
    and step:  $\bigwedge i. i \leq k \implies P\ i \implies P\ (i - 1)$ 
  shows  $P\ i$ 
proof -
  have  $\bigwedge i::nat. n = k - i \implies i \leq k \implies P\ i$  for n
  proof (induct n)
    case 0
    then have  $i = k$  by arith
    with base show  $P\ i$  by simp
  next
    case (Suc n)
    then have  $n = (k - (i + 1))$  by arith
    moreover have  $k: i + 1 \leq k$  using Suc.prems by arith
    ultimately have  $P\ (i + 1)$  by (rule Suc.hyps)
    from step[OF k this] show ?case by simp
  qed
  with le show ?thesis by fast
qed

```

2.1 Definition of carry received at position k

When adding two numbers m and n , the carry is *introduced* at position 1 but is *received* at position 2. The function below accounts for the latter case.

$$\begin{array}{r}
 \text{k: } 6 \ 5 \ 4 \ 3 \ 2 \ 1 \ 0 \\
 \text{c: } \qquad \qquad \qquad 1 \\
 \text{---} \\
 \text{m: } \quad 1 \ 0 \ 1 \ 0 \ 1 \ 0 \\
 \text{n: } \qquad \qquad \qquad 1 \ 1 \\
 \text{-----} \\
 \text{m + n: } 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0
 \end{array}$$

definition *bin-carry* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
bin-carry $a \ b \ k = (a \bmod 2^k + b \bmod 2^k) \text{ div } 2^k$

Carry in the subtraction of two natural numbers

definition *bin-narry* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
bin-narry $a \ b \ k = (\text{if } b \bmod 2^k > a \bmod 2^k \text{ then } 1 \text{ else } 0)$

Equivalent definition

definition *bin-narry2* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
bin-narry2 $a \ b \ k = ((2^k + a \bmod 2^k - b \bmod 2^k) \text{ div } 2^k + 1) \bmod 2$

lemma *bin-narry-equiv*: $\text{bin-narry } a \ b \ c = \text{bin-narry2 } a \ b \ c$

apply (*auto simp add: bin-narry-def bin-narry2-def*)

subgoal by (*smt add.commute div-less dvd-0-right even-Suc le-add-diff-inverse2 less-add-eq-less*)

mod-greater-zero-iff-not-dvd neq0-conv not-mod2-eq-Suc-0-eq-0 order-le-less zero-less-diff

zero-less-numeral zero-less-power)

subgoal by (*simp add: le-div-geq less-imp-diff-less*)

done

2.2 Properties of carries

lemma *div-sub*:

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

shows $(a - b) \text{ div } c = (\text{if } (a \bmod c < b \bmod c) \text{ then } a \text{ div } c - b \text{ div } c - 1 \text{ else } a \text{ div } c - b \text{ div } c)$

proof–

consider (*alb*) $a < b$ | (*ageb*) $a \geq b$ **by** *linarith*

then show *?thesis*

proof *cases*

case *alb*

then show *?thesis* **using** *div-le-mono* **by** *auto*

next

case *ageb*

```

obtain a1 a2 where a1-def: a1 = a div c and a2-def: a2 = a mod c and
a-def: a=a1*c+a2
using mod-div-decomp by blast
obtain b1 b2 where b1-def: b1 = b div c and b2-def: b2 = b mod c and b-def:
b=b1*c+b2
using mod-div-decomp by blast
have a1geb1: a1 ≥ b1 using ageb a1-def b1-def using div-le-mono by blast
show ?thesis
proof(cases c=0)
  assume c=0
  then show ?thesis by simp
next
assume cneq0: c ≠ 0
then show ?thesis
proof(cases a2 < b2)
  assume a2lb2: a2 < b2
  then show ?thesis
  proof(cases a1=b1)
    case True
    then show ?thesis using ageb a2lb2 a-def b-def by force
  next
  assume ¬(a1=b1)
  hence a1gb1: a1 > b1 using a1geb1 by auto
  have boundc: a2+c-b2 < c using a2lb2 cneq0 by linarith
  have a-b = (a1 - b1) * c + a2 - b2
    using a-def b-def a1geb1 nat-diff-add-eq1[of b1 a1 c a2 b2] by auto
  also have ... = (a1 - b1 - 1 + 1) * c + a2 - b2
    using a1gb1 Suc-diff-Suc[of b1 a1] by auto
  also have ... = (a1 - b1 - 1) * c + (a2 + c - b2)
    using div-eq-0-iff[of b2 c] mod-div-trivial[of b c] b2-def by force
  finally have (a-b) div c = a1 - b1 - 1 + (a2 + c - b2) div c
    using a-def b-def cneq0 by auto
  then show ?thesis
    using boundc div-less by (simp add: a1-def a2-def b1-def b2-def)
  qed
next
assume a2geb2: ¬ a2 < b2
then have (a - b) div c = ((a1 - b1) * c + (a2 - b2)) div c
  using a1geb1 a-def b-def nat-diff-add-eq1 by auto
then show ?thesis using a2geb2 div-add1-eq[of (a1-b1)*c a2-b2 c]
  by(auto simp add: b2-def a2-def a1-def b1-def less-imp-diff-less)
qed
qed
qed
qed

```

lemma dif-digit-formula: $a \geq b \longrightarrow (a - b)_k = (a_k + b_k + \text{bin-narry } a \ b \ k) \text{ mod } 2$

proof -

```

{
  presume asm:  $a \geq b \wedge a \bmod 2^k < b \bmod 2^k$ 
  then have  $\text{Suc}((a - b) \text{div } 2^k) = a \text{div } 2^k - b \text{div } 2^k$ 
    by (smt Nat.add-diff-assoc One-nat-def Suc-pred add commute diff-is-0-eq
div-add-self1
div-le-mono div-sub mod-add-self1 nat-le-linear neq0-conv plus-1-eq-Suc
power-not-zero
zero-neq-numeral)
  then have  $(a - b) \text{div } 2^k \bmod 2 = \text{Suc} (a \text{div } 2^k \bmod 2 + b \text{div } 2^k$ 
mod 2) mod 2
    by (smt diff-is-0-eq even-Suc even-diff-nat even-iff-mod-2-eq-zero le-less
mod-add-eq
nat.simps(3) not-mod-2-eq-1-eq-0)
}
moreover
{
  presume asm2:  $\neg a \bmod 2^k < b \bmod 2^k \wedge b \leq a$ 
  then have  $(a - b) \text{div } 2^k \bmod 2 = (a \text{div } 2^k \bmod 2 + b \text{div } 2^k \bmod$ 
2) mod 2
    using div-sub[of b 2^k a] div-le-mono even-add even-iff-mod-2-eq-zero
le-add-diff-inverse2[of b div 2^k a div 2^k] mod-mod-trivial[of - 2]
not-less[of a mod 2^k b mod 2^k] not-mod-2-eq-1-eq-0 div-sub by smt
}

ultimately show ?thesis
  by (auto simp add: bin-narry-def nth-bit-def)
qed

```

lemma *dif-narry-formula*:

$a \geq b \longrightarrow \text{bin-narry } a \ b \ (k + 1) = (\text{if } (a \text{div } 2^k < b \text{div } 2^k + \text{bin-narry } a \ b \ k) \text{ then } 1 \text{ else } 0)$

proof –

```

{
  presume a1:  $a \bmod (2 * 2^k) < b \bmod (2 * 2^k)$ 
  presume a2:  $\neg a \text{div } 2^k \bmod 2 < \text{Suc} (b \text{div } 2^k \bmod 2)$ 
  have f3:  $2^k \neq (0::\text{nat})$ 
    by simp
  have f4:  $a \text{div } 2^k \bmod 2 = 1$ 
    using a2 by (meson le-less-trans mod2-eq-if mod-greater-zero-iff-not-dvd
not-less
zero-less-Suc)
  then have  $b \bmod (2 * 2^k) = b \bmod 2^k$ 
    using a2 by (metis (no-types) One-nat-def le-simps(3) mod-less-divisor
mod-mult2-eq
mult.left-neutral neq0-conv not-less semiring-normalization-rules(7))
  then have False
    using f4 f3 a1 by (metis One-nat-def add commute div-add-self1 div-le-mono
less-imp-le)
}

```

```

    mod-div-trivial mod-mult2-eq mult.left-neutral not-less plus-1-eq-Suc
    semiring-normalization-rules(7) zero-less-Suc)
  }
  moreover
  {
    presume a1:  $\neg a \bmod 2^k < b \bmod 2^k$ 
    presume a2:  $a \bmod (2 * 2^k) < b \bmod (2 * 2^k)$ 
    presume a3:  $\neg a \operatorname{div} 2^k \bmod 2 < b \operatorname{div} 2^k \bmod 2$ 
    presume a4:  $b \leq a$ 
    have f6:  $a \bmod 2^{\operatorname{Suc} k} < b \bmod 2^{\operatorname{Suc} k}$ 
    using a2 by simp
    obtain nn :: nat  $\Rightarrow$  nat  $\Rightarrow$  nat where f7:  $b + nn \ a \ b = a$  using a4
    le-add-diff-inverse by auto
    have (a div 2k - b div 2k) div 2 = a div 2k div 2 - b div 2k div 2
    using a3 div-sub by presburger
    then have f8:  $(a - b) \operatorname{div} 2^{\operatorname{Suc} k} = a \operatorname{div} 2^{\operatorname{Suc} k} - b \operatorname{div} 2^{\operatorname{Suc} k}$ 
    using a1 by (metis (no-types) div-mult2-eq div-sub power-Suc power-commutes)
    have f9:  $\forall n \ na. \operatorname{Suc} (na \operatorname{div} n) = (n + na) \operatorname{div} n \vee 0 = n$ 
    by (metis (no-types) add-Suc-right add-cancel-left-right div-add-self1 lessI
        less-Suc-eq-0-disj less-one zero-neq-one)
    then have  $\forall n \ na \ nb. (na + nb - n) \operatorname{div} na = \operatorname{Suc} (nb \operatorname{div} na) - n \operatorname{div} na -$ 
    1  $\vee$ 
     $\neg (na + nb) \bmod na < n \bmod na \vee 0 = na$  by (metis (no-types) div-sub)
    then have f10:  $\forall n \ na \ nb. \neg (nb :: nat) \bmod na < n \bmod na \vee nb \operatorname{div} na - n$ 
    div na
    =  $(na + nb - n) \operatorname{div} na \vee 0 = na$ 
    by (metis (no-types) diff-Suc-Suc diff-commute diff-diff-left mod-add-self1
        plus-1-eq-Suc)
    have  $\forall n. \operatorname{Suc} n \neq n$  by linarith
    then have (0 :: nat) = 2Suc k
    using f10 f9 f8 f7 f6 a4 by (metis add-diff-cancel-left' add-diff-assoc)
    then have False
    by simp
  }

  ultimately show ?thesis
  using bin-narry-def apply (auto simp add: nth-bit-def)
  subgoal by (smt add-0-left add-less-cancel-left divmod-digit-0(2) le-less-trans
    mod-less-divisor
    mod-mult2-eq mult-zero-right nat-neq-iff not-less not-mod2-eq-Suc-0-eq-0
    semiring-normalization-rules(7) zero-less-numeral zero-less-power)
  subgoal by (smt One-nat-def add.left-neutral divmod-digit-0(1) le-less-trans
    less-imp-le
    mod-less-divisor mod-mult2-eq mod-mult-self1-is-0 mult-zero-right not-less
    not-mod2-eq-Suc-0-eq-0 not-one-le-zero semiring-normalization-rules(7)
    zero-less-numeral
    zero-less-power)
  done
qed

```

lemma *sum-digit-formula*: $(a + b)_i k = (a_i k + b_i k + \text{bin-carry } a \ b \ k) \text{ mod } 2$
by (*simp add: bin-carry-def nth-bit-def*) (*metis div-add1-eq mod-add-eq*)

lemma *sum-carry-formula*: $\text{bin-carry } a \ b \ (k + 1) = (a_i k + b_i k + \text{bin-carry } a \ b \ k)$
div 2
by (*simp add: bin-carry-def nth-bit-def*)
(*smt div-mult2-eq div-mult-self4 mod-mult2-eq power-not-zero semiring-normalization-rules(20)*)
semiring-normalization-rules(34) semiring-normalization-rules(7) zero-neq-numeral)

lemma *bin-carry-bounded*:
shows $\text{bin-carry } a \ b \ k = \text{bin-carry } a \ b \ k \text{ mod } 2$
proof–
have $a \text{ mod } 2^k < 2^k$ **by** *simp*
moreover have $b \text{ mod } 2^k < 2^k$ **by** *simp*
ultimately have $(a \text{ mod } 2^k + b \text{ mod } 2^k) < 2^{(\text{Suc } k)}$ **by** (*simp add:*
mult-2 add-strict-mono)
then have $(a \text{ mod } 2^k + b \text{ mod } 2^k) \text{ div } 2^k \leq 1$ **using** *less-mult-imp-div-less*
by *force*
then have $\text{bin-carry } a \ b \ k \leq 1$ **using** *div-le-mono bin-carry-def* **by** *fastforce*
then show *?thesis* **by** *auto*
qed

lemma *carry-bounded*: $\text{bin-carry } a \ b \ k \leq 1$
using *bin-carry-bounded not-mod-2-eq-1-eq-0[of bin-carry a b k]* **by** *auto*

lemma *no-carry*:
 $(\forall r < n. ((\text{nth-bit } a \ r) + (\text{nth-bit } b \ r) \leq 1)) \implies$
 $(\text{nth-bit } (a + b) \ n) = (\text{nth-bit } a \ n + \text{nth-bit } b \ n) \text{ mod } 2$
(is *?P* \implies *?Q n*)
proof (*rule ccontr*)
assume *p*: *?P*
assume *nq*: $\neg ?Q \ n$
then obtain *k* **where** *k1*: $\neg ?Q \ k$ **and** *k2*: $\forall r < k. ?Q \ r$ **by** (*auto dest: obtain-smallest*)

have *c1*: $1 = \text{bin-carry } a \ b \ k$
using *k1 sum-digit-formula bin-carry-bounded*
by *auto (metis add.commute not-mod2-eq-Suc-0-eq-0 plus-nat.add-0)*

have $\text{bin-carry } a \ b \ (k-1) = 0$ **using** *sum-digit-formula*
by (*metis bin-carry-bounded bin-carry-def diff-is-0-eq' diff-less div-by-1 even-add*
even-iff-mod-2-eq-zero k2 less-numeral-extra(1) mod-by-1 neq0-conv
nth-bit-bounded
power-0)

moreover have $a_i (k-1) + b_i (k-1) < 1$
by (*smt add.right-neutral c1 calculation diff-le-self k2 leI le-add-diff-inverse2*
le-less-trans)

*mod-by-1 mod-if nat-less-le nq one-div-two-eq-zero one-neq-zero p
sum-carry-formula)*

ultimately have $0 = \text{bin-carry } a \ b \ k$ **using** $k2$ *sum-carry-formula*
by *auto* (*metis Suc-eq-plus1-left add-diff-inverse-nat less-imp-diff-less mod-0
mod-Suc*
mod-add-self1 mod-div-trivial mod-less n-not-Suc-n plus-nat.add-0)

then show *False* **using** *c1* **by** *auto*
qed

lemma *no-carry-mult-equiv*: $(\forall k. \text{nth-bit } a \ k * \text{nth-bit } b \ k = 0) \longleftrightarrow (\forall k. \text{bin-carry } a \ b \ k = 0)$

(**is** $?P \longleftrightarrow ?Q$)

proof

assume P : $?P$

{

fix k

from P **have** $\text{bin-carry } a \ b \ k = 0$

proof (*induction k*)

case 0

then show $?case$ **using** *bin-carry-def* **by** (*simp*)

next

case (*Suc k*)

then show $?case$ **using** *sum-carry-formula P*

by (*metis One-nat-def Suc-eq-plus1 add.right-neutral div-less lessI*

mult-is-0 not-mod-2-eq-0-eq-1 nth-bit-def numeral-2-eq-2 zero-less-Suc)

qed

}

then show $?Q$ **by** *auto*

next

assume Q : $?Q$

{

fix k

from Q **have** $a \ j \ k * b \ j \ k = 0$

proof (*induction k*)

case 0

then show $?case$ **using** *bin-carry-def nth-bit-def*

by *simp* (*metis add-self-div-2 not-mod2-eq-Suc-0-eq-0 power-one-right*)

next

case (*Suc k*)

then show $?case$

using *nth-bit-def sum-carry-formula*

by *simp* (*metis One-nat-def add.right-neutral add-self-div-2 not-mod-2-eq-1-eq-0*

power-Suc)+

qed

}

then show $?P$ **by** *auto*

qed

```

lemma carry-digit-impl: bin-carry a b k ≠ 0 ⇒ ∃ r < k. a ⌋ r + b ⌋ r = 2
proof(rule ccontr)
  assume ¬ (∃ r < k. a ⌋ r + b ⌋ r = 2)
  hence bound: ∀ r < k. a ⌋ r + b ⌋ r ≤ 1 using nth-bit-def by auto
  assume bk: bin-carry a b k ≠ 0
  hence base: bin-carry a b k = 1 using carry-bounded le-less[of bin-carry a b k 1]
by auto
  have step: i ≤ k ⇒ bin-carry a b i = 1 ⇒ bin-carry a b (i - 1) = 1 for i
  proof(rule ccontr)
    assume ik: i ≤ k
    assume carry: bin-carry a b i = 1
    assume bin-carry a b (i - 1) ≠ 1
    hence bin-carry a b (i - 1) = 0 using bin-carry-bounded not-mod-2-eq-1-eq-0[of
bin-carry a b (i - 1)] by auto
    then show False using ik carry bound sum-carry-formula[of a b i - 1]
    apply simp
    by (metis Suc-eq-plus1 Suc-pred add-lessD1 bot-nat-0.not-eq-extremum carry
diff-is-0-eq' div-le-mono le-eq-less-or-eq less-add-one one-div-two-eq-zero)
    qed
  have ∀ i ≤ k. bin-carry a b i = 1 using rev-induct[where ?P = λc. (bin-carry a b c
= 1)] step base by blast
  moreover have bin-carry a b 0 = 0 using bin-carry-def by simp
  ultimately show False by auto
qed

```

```

end
theory Binary-Operations
  imports Bits-Digits Carries
begin

```

3 Digit-wise Operations

3.1 Binary AND

```

fun bitAND-nat :: nat ⇒ nat ⇒ nat (infix && 64) where
  0 && - = 0 |
  m && n = 2 * ((m div 2) && (n div 2)) + (m mod 2) * (n mod 2)

```

```

lemma bitAND-zero[simp]: n = 0 ⇒ m && n = 0
by (induct m n rule: bitAND-nat.induct, auto)

```

```

lemma bitAND-1: a && 1 = (a mod 2)
by (induction a; auto)

```


lemma *bitAND-rec*: $m \&\& n = 2 * ((m \text{ div } 2) \&\& (n \text{ div } 2)) + (m \text{ mod } 2) * (n \text{ mod } 2)$
by (*cases m; simp-all*)

lemma *bitAND-commutes*: $m \&\& n = n \&\& m$
by (*induct m n rule: bitAND-nat.induct, simp*) (*metis bitAND-rec mult.commute*)

lemma *nth-digit-0*: $x \leq 1 \implies \text{nth-bit } x \ 0 = x$ **by** (*simp add: nth-bit-def*)

lemma *bitAND-zeroone*: $a \leq 1 \implies b \leq 1 \implies a \&\& b \leq 1$
using *nth-bit-def nth-digit-0 nat-le-linear bitAND-nat.elims*
by (*metis (no-types, lifting) One-nat-def add.left-neutral bitAND-zero div-less le-zero-eq lessI mult.right-neutral mult-0-right not-mod2-eq-Suc-0-eq-0 numeral-2-eq-2*)

lemma *aux1-bitAND-digit-mult*:
fixes $a \ b \ c :: \text{nat}$
shows $k > 0 \wedge a \text{ mod } 2 = 0 \wedge b \leq 1 \implies (a + b) \text{ div } 2^k = a \text{ div } 2^k$
by (*induction k, auto*)
(*metis One-nat-def add-cancel-left-right div-mult2-eq even-succ-div-two le-0-eq le-Suc-eq*)

lemma *bitAND-digit-mult*: $(\text{nth-bit } (a \&\& b) \ k) = (\text{nth-bit } a \ k) * (\text{nth-bit } b \ k)$
proof(*induction k arbitrary: a b*)
case 0
show *?case*
using *nth-bit-def*
by *auto* (*metis (no-types, opaque-lifting) Groups.add-ac(2) bitAND-rec mod-mod-trivial mod-mult-self2 mult-numeral-1-right mult-zero-right not-mod-2-eq-1-eq-0 numeral-One*)
next
case (*Suc k*)
have $\text{nth-bit } (a \&\& b) \ (\text{Suc } k) = (2 * (a \text{ div } 2 \&\& b \text{ div } 2) + a \text{ mod } 2 * (b \text{ mod } 2)) \text{ div } 2 \wedge (\text{Suc } k) \text{ mod } 2$
using *bitAND-rec* **by** (*metis nth-bit-def*)
moreover **have** $(a \text{ mod } 2) * (b \text{ mod } 2) < (2 \wedge \text{Suc}(k))$
by (*metis One-nat-def lessI mult-numeral-1-right mult-zero-right not-mod-2-eq-1-eq-0 numeral-2-eq-2 numeral-One power-gt1 zero-less-numeral zero-less-power*)
ultimately **have** $\text{nth-bit } (a \&\& b) \ (\text{Suc } k) = (2 * (a \text{ div } 2 \&\& b \text{ div } 2)) \text{ div } 2 \wedge (\text{Suc } k) \text{ mod } 2$
using *aux1-bitAND-digit-mult*
by (*metis le-numeral-extra(1) le-numeral-extra(4) mod-mult-self1-is-0 mult-numeral-1-right mult-zero-right not-mod-2-eq-1-eq-0 numeral-One zero-less-Suc*)
then **have** $\text{nth-bit } (a \&\& b) \ (\text{Suc } k) = (\text{nth-bit } (a \text{ div } 2 \&\& b \text{ div } 2) \ k)$
by (*auto simp add: nth-bit-def*)

then have $\text{nth-bit } (a \ \&\& \ b) \ (\text{Suc } k) = (\text{nth-bit } (a \ \text{div } 2) \ k) * (\text{nth-bit } (b \ \text{div } 2) \ k)$
using *Suc*
by *presburger*

then show *?case*
by (*metis div-mult2-eq nth-bit-def power-Suc*)
qed

lemma *bitAND-single-bit-mult-equiv*: $a \leq 1 \implies b \leq 1 \implies a * b = a \ \&\& \ b$
using *bitAND-digit-mult[of a b 0] bitAND-zeroone* **by** (*auto simp: nth-digit-0*)

lemma *bitAND-mult-equiv*:
 $(\forall k. (\text{nth-bit } c \ k) = (\text{nth-bit } a \ k) * (\text{nth-bit } b \ k)) \longleftrightarrow c = a \ \&\& \ b$ (**is** *?P* \longleftrightarrow *?Q*)

proof
assume *?Q*
then show *?P* **using** *bitAND-digit-mult* **by** *simp*
next
assume *?P*
then show *?Q* **using** *bitAND-digit-mult digit-wise-equiv* **by** *presburger*
qed

lemma *bitAND-linear*:
fixes $k::\text{nat}$
shows $(b < 2^k) \wedge (d < 2^k) \implies (a * 2^k + b) \ \&\& \ (c * 2^k + d) = (a \ \&\& \ c) * 2^k + (b \ \&\& \ d)$

proof(*induction k arbitrary: a b c d*)

case *0*
then show *?case* **by** *simp*

next
case (*Suc k*)
define m **where** $m = a * 2^{\text{Suc } k} + b$
define n **where** $n = c * 2^{\text{Suc } k} + d$

have $m \ \&\& \ n = 2 * (\text{bitAND-nat } (m \ \text{div } 2) \ (n \ \text{div } 2)) + (m \ \text{mod } 2) * (n \ \text{mod } 2)$
using *bitAND-rec*
by *blast*

moreover have $d \ \text{mod } 2 = n \ \text{mod } 2 \wedge b \ \text{mod } 2 = m \ \text{mod } 2$
by (*metis m-def n-def add commute mod-mult-self2 power-Suc semiring-normalization-rules(19)*)

ultimately have $f0:m \ \&\& \ n$
 $= 2 * ((a * 2^k + (b \ \text{div } 2)) \ \&\& \ (c * 2^k + (d \ \text{div } 2))) + (b \ \text{mod } 2) * (d \ \text{mod } 2)$
by (*metis add commute div-mult-self2 m-def n-def power-Suc semiring-normalization-rules(19) zero-neq-numeral*)

have $b \ \text{div } 2 < (2^k) \wedge d \ \text{div } 2 < (2^k)$

```

    using Suc.prem
    by auto

  then have f1:m && n
    = ((a && c) * 2^(Suc k)) + 2 * ((b div 2) && (d div 2)) + (b mod
2) * (d mod 2)
    using f0 Suc.IH
    by simp

  have b && d = 2 * ((b div 2) && (d div 2)) + (b mod 2) * (d mod 2)
    using bitAND-rec
    by blast

  then show ?case
    using f1
    by (auto simp add: m-def n-def)
qed

```

3.2 Binary orthogonality

cf. [1] section 2.6.1 on "Binary orthogonality"

The following definition differs slightly from the one in the paper. However, we later prove the equivalence of the two definitions.

```

fun orthogonal :: nat => nat => bool (infix  $\perp$  49) where
  (orthogonal a b) = (a && b = 0)

```

```

lemma ortho-mult-equiv: a  $\perp$  b  $\longleftrightarrow$  ( $\forall k. (nth-bit a k) * (nth-bit b k) = 0$ ) (is ?P
 $\longleftrightarrow$  ?Q)

```

proof

```

  assume ?P
  then show ?Q using bitAND-digit-mult nth-bit-def by (metis div-0 mod-0 or-
thogonal.simps)

```

next

```

  assume ?Q
  then show ?P using bitAND-mult-equiv nth-bit-def by (metis div-0 mod-0
orthogonal.simps)

```

qed

```

lemma aux1-1-digit-lt-linear:

```

```

  assumes b < 2^r k  $\geq$  r
  shows bin-carry (a*2^r) b k = 0

```

proof–

```

  have b < 2^r  $\longrightarrow$  (a*2^r)  $\perp$  b
  proof(induct a b rule: bitAND-nat.induct)
    case (1 uu)
    then show ?case by simp
  next
    case (2 v n)

```

show *?case* **apply** *auto* **using** *bitAND-linear[of n r 0 0 Suc(v)] bitAND-commutes*
by *auto*
qed
then show *?thesis* **using** *ortho-mult-equiv no-carry-mult-equiv assms(1)* **by** *auto*
qed

lemma *aux1-digit-lt-linear*:
assumes $b < 2^r$ **and** $k \geq r$
shows $(a * 2^r + b) \dot{\div} k = (a * 2^r) \dot{\div} k$
proof–
have $b \text{ div } 2^k = 0$ **using** *assms* **by** (*simp add: order.strict-trans2*)
moreover have $(a * 2^r \text{ mod } 2^k + b \text{ mod } 2^k) \text{ div } 2^k = 0$ **using** *assms*
proof–
have *bin-carry* $(a * 2^r) b k = 0$ **using** *assms aux1-1-digit-lt-linear* **by** *auto*
then show *?thesis* **using** *assms* **by** (*auto simp add: bin-carry-def*)
qed
ultimately show *?thesis*
by (*auto simp add: nth-bit-def div-add1-eq[of a*2^r b 2^k]*)
qed

lemma *aux-digit-shift*: $(a * 2^t) \dot{\div} (l+t) = a \dot{\div} l$
using *nth-bit-def*
by (*induct l; auto*)
(smt div-mult2-eq mult commute nonzero-mult-div-cancel-right power-add power-not-zero zero-neq-numeral)

lemma *aux-digit-lt-linear*:
assumes $b < (2::nat)^t$
assumes $d < (2::nat)^t$
shows $(a * 2^t + b) \dot{\div} k \leq (c * 2^t + d) \dot{\div} k \iff ((a * 2^t) \dot{\div} k \leq (c * 2^t) \dot{\div} k \wedge b \dot{\div} k \leq d \dot{\div} k)$
proof (*cases k < t*)
case *True*
from *True* **have** $(a * 2^t + b) \dot{\div} k = b \dot{\div} k$
using *aux2-digit-sum-repr assms(1)* **by** *auto*
moreover from *True* **have** $(c * 2^t + d) \dot{\div} k = d \dot{\div} k$
using *aux2-digit-sum-repr assms(2)* **by** *auto*
moreover from *True* **have** $(a * 2^t) \dot{\div} k = 0$
using *aux2-digit-sum-repr[of 0] nth-bit-def* **by** *auto*
ultimately show *?thesis*
using *aux2-digit-sum-repr assms* **by** *auto*
next
case *False*
from *False* **have** $(a * 2^t + b) \dot{\div} k = (a * 2^t) \dot{\div} k$
using *aux1-digit-lt-linear assms(1)* **by** *auto*
moreover from *False* **have** $(c * 2^t + d) \dot{\div} k = (c * 2^t) \dot{\div} k$ **using** *aux1-digit-lt-linear assms(2)* **by** *auto*
moreover from *False* **have** $b \dot{\div} k = 0$
using *aux1-digit-lt-linear[of - - 0] nth-bit-def assms(1)* **by** *auto*

ultimately show *?thesis* by *auto*
qed

lemma *aux2-digit-lt-linear*:
fixes $a\ b\ c\ d\ t\ l :: \text{nat}$
shows $\exists k. (a * 2^t) \dot{\vdash} k \leq (c * 2^t) \dot{\vdash} k \longrightarrow a \dot{\vdash} l \leq c \dot{\vdash} l$
proof –
define k where $k = l + t$
have $(a * 2^t) \dot{\vdash} k = a \dot{\vdash} l$ **using** *nth-bit-def k-def*
using *aux-digit-shift* by *auto*
moreover **have** $(c * 2^t) \dot{\vdash} k = c \dot{\vdash} l$ **using** *nth-bit-def k-def*
using *aux-digit-shift* by *auto*
ultimately show *?thesis* by *metis*
qed

lemma *aux3-digit-lt-linear*:
fixes $a\ b\ c\ d\ t\ k :: \text{nat}$
shows $\exists l. a \dot{\vdash} l \leq c \dot{\vdash} l \longrightarrow (a * 2^t) \dot{\vdash} k \leq (c * 2^t) \dot{\vdash} k$
proof (*cases* $k < t$)
case *True*
hence $(a * 2^t) \dot{\vdash} k = 0$
using *aux2-digit-sum-repr[of 0] nth-bit-def* by *auto*
then show *?thesis* by *auto*
next
case *False*
define l where $l = k - t$
hence $k: k = l + t$ **using** *False* by *auto*
have $(a * 2^t) \dot{\vdash} k = a \dot{\vdash} l$ **using** *nth-bit-def l-def*
using *aux-digit-shift k* by *auto*
moreover **have** $(c * 2^t) \dot{\vdash} k = c \dot{\vdash} l$ **using** *nth-bit-def l-def*
using *aux-digit-shift k* by *auto*
ultimately show *?thesis* by *auto*
qed

lemma *digit-lt-linear*:
fixes $a\ b\ c\ d\ t :: \text{nat}$
assumes $b: b < (2::\text{nat})^t$
assumes $d: d < (2::\text{nat})^t$
shows $(\forall k. (a * 2^t + b) \dot{\vdash} k \leq (c * 2^t + d) \dot{\vdash} k) \longleftrightarrow (\forall l. a \dot{\vdash} l \leq c \dot{\vdash} l \wedge b \dot{\vdash} l \leq d \dot{\vdash} l)$
proof –
have *shift*: $(\forall k. (a * 2^t) \dot{\vdash} k \leq (c * 2^t) \dot{\vdash} k) \longleftrightarrow (\forall l. a \dot{\vdash} l \leq c \dot{\vdash} l)$ (**is** $?P \longleftrightarrow ?Q$)
proof
assume $P: ?P$
show $?Q$ **using** P *aux2-digit-lt-linear* by *auto*
next
assume $Q: ?Q$
show $?P$ **using** Q *aux3-digit-lt-linear* by *auto*

qed

have *main*: $(\forall k. (a * 2^t + b) \dot{\vdash} k \leq (c * 2^t + d) \dot{\vdash} k \iff ((a * 2^t) \dot{\vdash} k \leq (c * 2^t) \dot{\vdash} k \wedge b \dot{\vdash} k \leq d \dot{\vdash} k))$
using *aux-digit-lt-linear* *b d* **by** *auto*

from *main shift* **show** *?thesis* **by** *auto*
qed

Sufficient bitwise (digitwise) condition for the non-strict standard order of natural numbers

lemma *digitwise-leq*:

assumes $b > 1$

shows $\forall t. \text{nth-digit } x \ t \ b \leq \text{nth-digit } y \ t \ b \implies x \leq y$

proof –

assume *asm*: $\forall t. \text{nth-digit } x \ t \ b \leq \text{nth-digit } y \ t \ b$

define *r* **where** $r \equiv (\text{if } x > y \text{ then } x \text{ else } y)$

have $x = (\sum k < x. (\text{nth-digit } x \ k \ b) * b^k)$

using *digit-gen-sum-repr-variant* $\langle b > 1 \rangle$ **by** *auto*

hence *x*: $x = (\sum k = 0..<r. (\text{nth-digit } x \ k \ b) * b^k)$

using *atLeast0LessThan* *r-def digit-gen-sum-index-variant* $\langle b > 1 \rangle$

by (*metis* (*full-types*) *linorder-neqE-nat*)

have $y = (\sum k < y. (\text{nth-digit } y \ k \ b) * b^k)$

using *digit-gen-sum-repr-variant* $\langle b > 1 \rangle$ **by** *auto*

hence *y*: $y = (\sum k = 0..<r. (\text{nth-digit } y \ k \ b) * b^k)$

using *atLeast0LessThan* *r-def digit-gen-sum-index-variant* $\langle b > 1 \rangle$ **by** *auto*

show *?thesis* **using** *asm* *x y*

sum-mono[of $\{0..<r\}$] $\lambda k. \text{nth-digit } x \ k \ b * b^k \ \lambda k. \text{nth-digit } y \ k \ b * b^k$]

by *auto*

qed

3.3 Binary masking

Preliminary result on the standard non-strict of natural numbers

lemma *bitwise-leq*: $(\forall k. a \dot{\vdash} k \leq b \dot{\vdash} k) \longrightarrow a \leq b$

using *digitwise-leq*[of 2] **by** (*simp* *add: nth-digit-base2-equiv*)

cf. [1] section 2.6.2 on "Binary Masking"

Again, the equivalence to the definition there will be proved in a later lemma.

fun *masks* :: *nat* => *nat* => *bool* (**infix** \preceq 49) **where**

masks 0 - = *True* |

masks *a b* = $((a \text{ div } 2 \preceq b \text{ div } 2) \wedge (a \text{ mod } 2 \leq b \text{ mod } 2))$

lemma *masks-substr*: $a \preceq b \implies (a \text{ div } (2^k) \preceq b \text{ div } (2^k))$

proof (*induction* *k*)

case 0

then **show** *?case* **by** *simp*

```

next
  case (Suc k)
  moreover
  {
    fix ka :: nat
    assume a1: a div 2 ^ ka ≤ b div 2 ^ ka

    have f2: ∀ n na nb. (nb::nat) div na div n = nb div n div na
      by (metis (no-types) div-mult2-eq semiring-normalization-rules(7))

    then have f3: ∀ n na nb nc.
      (nc div nb = 0 ∨ nc div 2 div nb ≤ na div 2 div n) ∨ ¬ nc div nb
    ≤ na div n
      by (metis (no-types) masks.elims(2))

    {
      assume ∃ n. a div n div 2 ^ ka ≠ 0 then have a div 2 ^ ka ≠ 0 using f2 by
      (metis div-0)
      then have a div 2 div 2 ^ ka ≤ b div 2 div 2 ^ ka using f3 a1 by meson
    }
    then have a div (2 * 2 ^ ka) ≤ b div (2 * 2 ^ ka)
      by (metis (no-types) div-mult2-eq masks.simps(1))
  }
  ultimately show ?case by simp
qed

lemma masks-digit-leq:(a ≤ b) ⇒ (nth-bit a k) ≤ (nth-bit b k)
proof (induction k arbitrary: a b)
  case 0
  then show ?case
    by (metis add-cancel-left-right bitAND-nat.elims div-by-1 le0 masks.simps(2)
    power-0 mod-mult-self1-is-0 mod-mult-self4 nth-bit-def)
next
  case (Suc k)
  then show ?case
    by (simp add: nth-bit-def)
    (metis div-mult2-eq masks-substr nth-bit-def pow.simps(1) power-numeral)
qed

lemma masks-leq-equiv:(a ≤ b) ⇔ (∀ k. (nth-bit a k) ≤ (nth-bit b k)) (is ?P ⇔
?Q)
proof
  assume ?P
  then show ?Q using masks-digit-leq by auto
next
  assume ?Q
  then show ?P using nth-bit-def
  proof (induct a b rule: masks.induct)

```

```

    case (1 uu)
    then show ?case by simp
  next
    case (2 v b)
    then show ?case by simp (metis drop-bit-Suc drop-bit-eq-div div-by-1 power.simps(1))
  qed
qed

```

lemma *masks-leq*: $a \preceq b \longrightarrow a \leq b$
using *masks-leq-equiv bitwise-leq* **by** *simp*

lemma *mask-linear*:

```

  fixes a b c d t :: nat
  assumes b: b < (2::nat) ^ t
  assumes d: d < (2::nat) ^ t
  shows ((a * 2 ^ t + b) ≤ c * 2 ^ t + d) ⟷ (a ≤ c ∧ b ≤ d) (is ?P ⟷ ?Q)
proof -
  have ?P ⟷ (∀ k. (a * 2 ^ t + b) i k ≤ (c * 2 ^ t + d) i k) using masks-leq-equiv
by auto
  also have ... ⟷ (∀ k. a i k ≤ c i k ∧ b i k ≤ d i k) using b d digit-lt-linear by
auto
  also have ... ⟷ a ≤ c ∧ b ≤ d using masks-leq-equiv by auto
  finally show ?thesis by auto
qed

```

lemma *aux1-lm0241-pow2-up-bound*: $(\exists (p::nat). (a::nat) < 2^{\wedge}(Suc\ p))$
by (*induction a*) (*use less-exp in fastforce*)+

lemma *aux2-lm0241-single-digit-binom*:

```

  assumes 1 ≥ (a::nat)
  assumes 1 ≥ (b::nat)
  shows ¬(a = 1 ∧ b = 1) ⟷ ((a + b) choose b) = 1 (is ?P ⟷ ?Q)
using assms(1) assms(2)
by (metis Suc-eq-plus1 add commute add-cancel-right-left add-eq-if
    binomial-n-0 choose-one le-add2 le-antisym zero-neq-one)

```

lemma *aux3-lm0241-binom-bounds*:

```

  assumes 1 ≥ (m::nat)
  assumes 1 ≥ (n::nat)
  shows 1 ≥ m choose n
using assms(1) assms(2) le-Suc-eq by auto

```

lemma *aux4-lm0241-prod-one*:

```

  fixes f::(nat ⇒ nat)
  assumes (∀ x. (1 ≥ f x))
  shows (∏ k ≤ n. (f k)) = 1 ⟶ (∀ k. k ≤ n ⟶ f k = 1) (is ?P ⟶ ?Q)
proof(rule ccontr)
  assume assm: ¬(?P ⟶ ?Q)
  hence f-zero: ∃ r. r ≤ n ∧ f r ≠ 1 by simp

```


then obtain r **where** $f r \neq 1$ **and** $r \leq n$ **by** *blast*
hence $f r = 0$ **using** *assms le-antisym not-less* **by** *blast*
hence *contr*: $(\prod k \leq n. f k) = 0$ **using** $\langle r \leq n \rangle$ **by** *auto*
then show *False* **using** *assm contr* **by** *simp*
qed

lemma *aux5-lm0241*:

$(\forall i. (nth-bit (a + b) i) \text{ choose } (nth-bit b i) = 1) \longrightarrow$
 $\neg(nth-bit a i = 1 \wedge nth-bit b i = 1)$
(is $?P \longrightarrow ?Q$ $i)$

proof(*rule ccontr*)

assume *assm*: $\neg(?P \longrightarrow ?Q i)$

hence $(\exists i. \neg ?Q i)$ **by** *blast*

then obtain i **where** *contr*: $\neg ?Q i$ **and** *i-minimal*: $(\forall j < i. ?Q j)$

using *obtain-smallest*[*of* $\langle \lambda i. \neg ?Q i \rangle$] **by** *auto*

hence $\forall j. j < i \longrightarrow nth-bit a j * nth-bit b j = 0$ **by** (*simp add: nth-bit-def*)

hence $\forall j. j < i \longrightarrow ((nth-bit a j = 0 \wedge nth-bit b j = 1) \vee$
 $(nth-bit a j = 1 \wedge nth-bit b j = 0) \vee$
 $(nth-bit a j = 0 \wedge nth-bit b j = 0))$

by (*auto simp add: nth-bit-def*)

hence $\forall j. j < i \longrightarrow nth-bit a j + nth-bit b j \leq 1$ **by** *auto*

hence *bin-carry* $a b i = 0$

using *no-carry* **by** (*metis contr add-self-mod-2 assm choose-one one-neq-zero*)

hence *f0*: $nth-bit (a + b) i = (nth-bit a i + nth-bit b i) \text{ mod } 2$

by(*auto simp add:sum-digit-formula*)

have $\dots = 0$ **using** *contr* **by** *auto*

hence $(nth-bit (a + b) i) \text{ choose } (nth-bit b i) = 0$ **using** *f0 contr* **by** *auto*

then show *False* **using** *assm* **by** *fastforce*

qed

end

References

- [1] Y. Matiyasevich. On Hilbert's tenth problem. In M. Lamoureux, editor, *PIMS Distinguished Chair Lectures*, volume 1. Pacific Institute for the Mathematical Sciences, 2000.