Internal Note #182
Computational Logic, Inc.
March 14, 1990

AN INTEGER LIBRARY FOR NQTHM

Matt Kaufmann[1]

Computational Logic, Inc.
1717 W. Sixth Street, Suite 290
Austin, TX 78703

kaufmann@cli.com

This note contains a brief summary of an integer library that has been created for nqthm. This discussion is far from thorough and complete[2], but I felt that a summary of some of the main ideas would be lots better than nothing at all. An integer library has in fact existed for some time at CLI, thanks to Bill Bevier and Matt Wilding (cf. [1]), and the library reported on in this note includes much of that library and owes an intellectual debt to it. The main new contribution has been to enrich the collection of useful metalemmas proved about the integers, and in this note I'll confine myself mainly, though not completely, to comments about these lemmas.

As CLI users would expect, this library can be loaded by using **`(note-lib "/usr/local/src/nqthm-libs/integers")`**. If you're not using pc-nqthm, then you should probably follow that with **`(load "/usr/local/src/nqthm-libs/integers")`** in order to get the compiled

[1]This research was supported in part by ONR Contract N00014-88-C-0454. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of Computational Logic, Inc., the Office of Naval Research or the U.S. Government.

[2]and there is quite surely room for improvement in the library too!

‘‘*1*’’ (executable) functions (especially, metafunctions). The library has been designed so that one can use it profitably without reading the rest of this note. The rest of the note may, however, give the user some ideas on how better to take advantage of the library. For example, it is suggested that one may want to experiment by enabling the rewrite rule ILESSP-TRICHOTOMY.

The first section below briefly summarizes the basic integer functions and some rewrite rules defined in this library. These are essentially unchanged from previous versions (though there are a couple of minor changes, notably in the definition of ILEQ). The second section states the metalemmas and describes the algorithms they implement, and suggests possible actions on the part of the user to adjust the set of metalemmas that are enabled. The final section has some miscellaneous caveats and suggestions.

Much of this library was created in an attempt to prove some challenge theorems suggested by Don Simon, who I thank for those problems, and a challenge problem given by Bill Pase that appears in the final section below.

1. Summary of basic integer functions and some basic rewrite rules

I’ll omit here the actual definitions of the basic integer functions. As mentioned above, these are essentially unchanged from previous versions; and of course they can be found in /usr/local/src/nqthm-libs/integers.events. Perhaps the greatest change is that now **`(ILEQ I J)`** is now defined to be **`(NOT (ILESSP J I))`**, rather than as **`(OR (ILESSP I J) (EQUAL I J))`**. Thus, like the rest of the integer functions, the value doesn’t change if **`FIX-INT`** is applied to the arguments. The basic functions defined in this library, which are defined in terms of natural number functions, have been put into the following theory:

```
(deftheory all-integer-defns
  (integerp fix-int izerop ilessp ileq iplus ineg idifference
   iabs itimes iquotient iremainder idiv imod iquo irem))
```

In accordance with the usual way arithmetic functions are defined in nqthm, these functions act as though non-integer arguments are coerced to 0. The history has been left in a state where integer functions (e.g. ILESSP, FIX-INT, IPLUS, ...) have been disabled, with three exceptions: IZEROP, ILEQ, and IDIFFERENCE. Therefore there are very few rules about IZEROP (but plenty of rules about FIX-INT), about ILEQ (but lots of rules about ILESSP), or about IDIFFERENCE (but many about IPLUS and INEG). If one wants to prove a basic integer fact by opening up functions that are normally disabled, an (ENABLE-THEORY INTEGER-DEFNS) hint should be given, where INTEGER-DEFNS is defined as follows.

```
(deftheory integer-defns
  (integerp fix-int ilessp iplus ineg iabs itimes
            iquotient iremainder idiv imod iquo irem))
```

Let us turn briefly to rewrite rules. There are rewrite rules expressing the commutativity and associativity of IPLUS and ITIMES. (Notice that it therefore follows that for the metalemmas, it suffices to deal with terms that are right-associated with respect to these operations.) Also included are distributive laws that push IPLUS out of ITIMES expressions and out of INEG expressions and that push INEG out of ITIMES expressions. Rules have been proved which state that 0 and 1 are the respective identities for integer addition (IPLUS) and integer multiplication (ITIMES). There are also a number of trivial ‘‘type’’ rules, e.g. IPLUS always returns an integer and implicitly applies FIX-INT to its arguments. Finally, there are rules for combining constants in IPLUS terms, so that for example **`(IPLUS 3 (IPLUS X 7))`** is rewritten to **`(IPLUS 10 X)`**, **`(EQUAL (IPLUS 10 X) (IPLUS Z 3))`** is rewritten to **`(EQUAL (IPLUS 7 X) (FIX-INT Z))`**, and **`(ILESSP (IPLUS 3 Z) (IPLUS 10 X))`** is rewritten to **`(NOT (ILESSP (IPLUS 6 X) Z))`**. Here are the lemmas used (besides commutativity) in these respective examples:

```
IPLUS-CONSTANTS
(EQUAL (IPLUS (ADD1 I) (IPLUS (ADD1 J) X))
       (IPLUS (PLUS (ADD1 I) (ADD1 J)) X))

CANCEL-CONSTANTS-EQUAL
(EQUAL (EQUAL (IPLUS (ADD1 I) X)
              (IPLUS (ADD1 J) Y))
       (IF (LESSP I J)
           (EQUAL (FIX-INT X)
                  (IPLUS (DIFFERENCE J I) Y))
           (EQUAL (IPLUS (DIFFERENCE I J) X)
                  (FIX-INT Y))))

CANCEL-CONSTANTS-ILESSP
(EQUAL (ILESSP (IPLUS (ADD1 I) X)
               (IPLUS (ADD1 J) Y))
       (IF (LESSP I J)
           (NOT (ILESSP (IPLUS (SUB1 (DIFFERENCE J I)) Y)
                        X))
           (ILESSP (IPLUS (DIFFERENCE I J) X)
                   Y)))
```

It had seemed that metalemmas might be needed for dealing with such constants, but in fact numeric constants are lower in the nqthm term order than terms with variables, so commutativity and associativity of IPLUS tend to push constants to the front of the sum.

2. The metalemmas and the algorithms they implement.

The reader is referred to the file /usr/local/src/nqthm-libs/integers.events for the definitions of the relevant metafunctions. Here I’ll simply state the metalemmas and make a few comments about them.

The first metalemma cancels pairs **`V`** and **`(INEG V)`** in a given sum. The idea of CANCEL-INEG is that it when it sees a sum **`(IPLUS x y)`**, it looks to see if the *formal negative* of **`x`** appears as a summand of **`y`**, where by the ‘‘formal negative’’ of **`x`** I mean **`(INEG x)`** unless **`x`** is of the form **`(INEG x1)`**, in which case I mean **`x1`**. If so, then CANCEL-INEG cancels that pair. Notice that since the rewriter works inside out, there is no harm in this algorithm's implicit assumption that **y** has already been simplified in this manner.

```
CORRECTNESS-OF-CANCEL-INEG:
(EQUAL (EVAL$ T X A)
       (EVAL$ T (CANCEL-INEG X) A)))
```

The next two metalemmas are for cancelling common summands on both sides of an equality or inequality (respectively). There are no particular surprises here. Notice that since the function symbol ILEQ is kept enabled, there is no need for a corresponding cancellation lemma for ILEQ.

```
CORRECTNESS-OF-CANCEL-IPLUS
(EQUAL (EVAL$ T X A)
       (EVAL$ T (CANCEL-IPLUS X) A)))

CORRECTNESS-OF-CANCEL-IPLUS-ILESSP
(EQUAL (EVAL$ T X A)
       (EVAL$ T (CANCEL-IPLUS-ILESSP X) A))
```

The next two metalemmas are for cancelling factors on both sides of an equality or inequality, respectively. The factors must be explicit, however. So for example, the first of these lemmas below will simplify the expression

```
     (EQUAL (ITIMES X Y) (ITIMES X Z))
```

to

```
     (IF (EQUAL (FIX-INT X) 0)
         T
         (EQUAL (FIX-INT Y) (FIX-INT Z)))
```

But it will not simplify the expression **`(EQUAL (IPLUS (ITIMES X Y) X) (IPLUS X (ITIMES X Z)))`**. Therefore, in fact I've proved stronger versions of these metalemmas, namely the third and fourth lemmas below (which strengthen the first and second, respectively). Because these last two are stronger, the first two are actually disabled in this library. If one finds these last two to be a nuisance for some reason, however (e.g. if they're too slow), it is of course easy enough to disable the last two and enable the first two. My preliminary impression at this point is that very little speed is sacrificed by having the latter pair enabled instead of the former.

```
CORRECTNESS-OF-CANCEL-ITIMES (disabled)
(EQUAL (EVAL$ T X A)
       (EVAL$ T (CANCEL-ITIMES X) A))

CORRECTNESS-OF-CANCEL-ITIMES-ILESSP (disabled)
(EQUAL (EVAL$ T X A)
       (EVAL$ T (CANCEL-ITIMES-ILESSP X) A))

CORRECTNESS-OF-CANCEL-ITIMES-FACTORS
(EQUAL (EVAL$ T X A)
       (EVAL$ T (CANCEL-ITIMES-FACTORS-EXPANDED X) A))

CORRECTNESS-OF-CANCEL-ITIMES-ILESSP-FACTORS
(EQUAL (EVAL$ T X A)
       (EVAL$ T (CANCEL-ITIMES-ILESSP-FACTORS X) A))
```

Remark. Notice that the metalemma CORRECTNESS-OF-CANCEL-ITIMES-FACTORS refers to the metafunction CANCEL-ITIMES-FACTORS-EXPANDED. There is in fact a function CANCEL-ITIMES-

FACTORS, but it seemed possible to gain efficiency by expanding all occurrences of AND and OR in the body of this function, since the executable counterparts of AND and OR are not lazy in nqthm. (Preliminary tests suggests only slight gains in efficiency, I must admit.) The Lisp macro NQTHM-MACROEXPAND, whose definition appears in a comment in /local/src/nqthm-libs/integers.events, was used to create the expanded version CANCEL-ITIMES-FACTORS-EXPANDED automatically. Somme other metafunction definitions have been similarly created by expansion. The following rule, followed by **`(disable cancel-itimes-factors-expanded)`**, eliminates the need to reason about the ‘‘expanded’’ function CANCEL-ITIMES-FACTORS-EXPANDED:

```
CANCEL-ITIMES-FACTORS-EXPANDED-CANCEL-ITIMES-FACTORS
(EQUAL (CANCEL-ITIMES-FACTORS-EXPANDED X)
       (CANCEL-ITIMES-FACTORS X))
```

[End of remark.]

Our final group of metalemmas takes care of some details not handled by the metalemmas above. The first of these, CORRECTNESS-OF-CANCEL-FACTORS-0, replaces a term that equates a product with 0 by the disjunction of the assertions saying that some factor equals 0. (More precisely, each disjunct says that **`FIX-INT`** applied to the factor equals 0.) The product can in fact be implicit, for as the following example shows, the function CANCEL-FACTORS-0 implicitly factors the expression completely.

```
>(r-loop)

Trace Mode: Off   Abbreviated Output Mode:  On
Type ? for help.
*(cancel-factors-0 ’(equal ’0 (itimes z (iplus x (itimes x y)))))
 ’(OR (EQUAL (FIX-INT Z) ’0)
      (OR (EQUAL (FIX-INT (IPLUS ’1 Y)) ’0)
          (EQUAL (FIX-INT X) ’0)))
*
```

Here then are the metalemmas that apply when a product is set equal to (or less than, or greater than) 0.

```
CORRECTNESS-OF-CANCEL-FACTORS-0
(EQUAL (EVAL$ T X A)
       (EVAL$ T (CANCEL-FACTORS-0 X) A))

CORRECTNESS-OF-CANCEL-FACTORS-ILESSP-0
(EQUAL (EVAL$ T X A)
       (EVAL$ T (CANCEL-FACTORS-ILESSP-0 X) A))
```

Finally, there are metalemmas for removing INEG terms from equations and inequalities. The desirability of performing such normalizations is illustrated by the following example:

```
(implies (equal (iplus x y)
                (iplus u v))
         (equal (fix-int x)
                (iplus u (idifference v y))))
```

Without the first metalemma below, the integer library fails to prove this obvious fact. However, that metalemma saves the day by replacing the term

```
     (EQUAL (FIX-INT X)
            (IPLUS U (IPLUS V (INEG Y))))
```

with the term

```
     (IF (INTEGERP (FIX-INT X))
         (EQUAL (IPLUS Y (FIX-INT X))
                (IPLUS U V))
         F)
```

which in turn simplifies to the hypothesis of the original implication. Here then are those final two metalemmas.

```
CORRECTNESS-OF-CANCEL-INEG-TERMS-FROM-EQUALITY
(EQUAL (EVAL$ T X A)
       (EVAL$ T (CANCEL-INEG-TERMS-FROM-EQUALITY-EXPANDED X) A))

CORRECTNESS-OF-CANCEL-INEG-TERMS-FROM-INEQUALITY
(EQUAL (EVAL$ T X A)
       (EVAL$ T (CANCEL-INEG-TERMS-FROM-INEQUALITY-EXPANDED X) A))
```

Note that these last two metalemmas do *not* subsume the first one, CORRECTNESS-OF-CANCEL-INEG. For these lemmas only apply to INEG terms that occur as a summand of the left or right side of an equation or inequality, while CORRECTNESS-OF-CANCEL-INEG cancels negatives in arbitrary sums.

3. Caveats

Here are a few things to watch out for when using this library. Please let me know if you find others.

First of all, the previous version of the integers library had more power for reasoning about terms in which integer functions are applied to arguments that are not known to be integers. For example, the following rewrite rule IPLUS-LEFT-ID is now disabled.

```
     (implies (not (integerp x))
              (equal (iplus x y)
                     (fix-int y)))
```

Here is a list of such rules about non-integers that have been left disabled: INEG-OF-NON-INTEGERP, IPLUS-LEFT-ID, IPLUS-RIGHT-ID, NOT-INTEGERP-IMPLIES-NOT-EQUAL-IPLUS, NOT-INTEGERP-IMPLIES-NOT-EQUAL-ITIMES. These rules have been disabled because one can pay a rather severe performance penalty otherwise. The example lemma FOUR-SQ shown below created a time triple of [0.1 62.4 0.2] on a Sun3/60 with the integer library loaded; but when the above rules were first enabled, the time triple was instead [0.1 84.9 0.2]. Two related rules that have also been left disabled to avoid backchaining are ITIMES-ZERO1 and ITIMES-ZERO2. Here is the former of those:

```
     (implies (equal (fix-int x) 0)
              (equal (itimes x y) 0))
```

With those two rules enabled in addition to the five listed above, the proof of this example took time

[0.1 797.6 0.3]! These rules aren't too important anyhow, given the metalemma CORRECTNESS-OF-CANCEL-FACTORS-0. A more difficult case is the rewrite rule SAME-FIX-INT-IMPLIES-NOT-ILESSP:

```
(implies (equal (fix-int x) (fix-int y))
         (not (ilessp x y)))
```

It wouldn't surprise me if it would be better to leave this rule enabled, but for now I'll leave it disabled. Another such potentially useful rewrite rule that I keep disabled is IZEROP-ILESSP-0-RELATIONSHIP:

```
(equal (equal (fix-int x) 0)
       (and (not (ilessp x 0))
            (not (ilessp 0 x))))
```

I also keep the rewrite rule ILESSP-TRICHOTOMY disabled, though I've found it helpful to enable this rule on occasion:

```
(implies (not (ilessp x y))
         (equal (ilessp y x)
                (not (equal (fix-int x) (fix-int y)))))
```

If both IZEROP-ILESSP-0-RELATIONSHIP and ILESSP-TRICHOTOMY are enabled together, one is in extreme danger of entering an infinite loop in the rewriter! In fact there is a note in integers.events suggesting that this rule gave me trouble in the proof of CORRECTNESS-OF-CANCEL-ITIMES-ILESSP-FACTORS.

Here is the test problem mentioned above. It was brought to my attention by Bill Pase of Odyssey Research Assoc.

```
(defn square (x)
  (itimes x x))

(defn four-squares (a b c d)
  (iplus (square a)
         (iplus (square b)
                (iplus (square c)
                       (square d)))))

(prove-lemma four-sq nil
  (equal (itimes (four-squares a b c d) (four-squares r s t0 u))
         (four-squares (iplus (itimes a r)
                              (iplus (itimes b s)
                                     (iplus (itimes c t0)
                                            (itimes d u))))
                       (iplus (itimes a s)
                              (iplus (ineg (itimes b r))
                                     (iplus (itimes c u)
                                            (ineg (itimes d t0)))))
                       (iplus (itimes a t0)
                              (iplus (ineg (itimes b u))
                                     (iplus (ineg (itimes c r))
                                            (itimes d s))))
                       (iplus (itimes a u)
                              (iplus (itimes b t0)
                                     (iplus (ineg (itimes c s))
                                            (ineg (itimes d r))))))))
```

Another thing to watch out for is the following problem that is not specific to the integer library in particular, but to all metalemmas: metalemmas only operate on so-called *tame terms*. The notion of ‘‘tame term’’ and a discussion of how it relates to metalemmas can be found in the discussion of metalemmas in [2], pp. 244-247. Very roughly speaking, a term is tame if has no occurrence of EVAL$ and related functions (actually some such occurrences are allowed), and if all defined function symbols occurring in the term have the property that their bodies are (recursively) tame. I've left a number of rewrite rules enabled that should sometimes help out when the metalemmas refuse to fire, but still, proof attempts for formulas about integers that contain non-tame terms are in more danger of failing than usual.

References

1. Bill Bevier, ‘‘A Library for Hardware Verification’’, Internal Note 57, Computational Logic, Inc., June 1988.

2. R. S. Boyer and J S. Moore, *A Computational Logic Handbook*, Academic Press, Boston, 1988.

AN INTEGER LIBRARY FOR NQTHM

Table of Contents