INTMUL, part 4. division

Johan G. F. Belinfante
2007 January 1

```
In[1]:= SetDirectory["l:"]; << goedel88.30a; << tools.m

        :Package Title: goedel88.30a        2006 December 30 at 2:45 p.m.

        It is now:  2007 Jan 1 at 4:54

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 December 17

        weightlimit = 40
```

summary

This notebook is concerned with integer division, and related matters. The first result derived is the fact that if a product of integers is zero, then one of the factors must be zero. (The integers form an integral domain.) The strategy used here is to reduce this basic theorem about integer multiplication to analogous results about mixed multiplication of integers by natural numbers, which are already available in the **GOEDEL** program. Using the distributive law, the following corollary is derived: left-multiplication by **x** is one-to-one if and only if **x** is nonzero. The proof presented here is mildly novel in that no unnecessary variables are introduced (but at a cost of some tricky simplification steps.) The integer division function is a restriction of **rotate[INTMUL]**. Here again some tricky simplifications are needed. The derivation presented here was closely patterned on a similar proof obtained in 2002 for the case of natural numbers.

relation of INTMUL to MIXMUL

One can relate **INTMUL** to **MIXMUL** using **PLUS**.

```
In[2]:= SubstTest[implies, and[member[s, binhom[v, w]], member[t, binhom[u, v]]],
          member[composite[s, t], binhom[u, w]],
          {s → composite[INTMUL, LEFT[x]], t → PLUS, u → NATADD, v → INTADD, w → INTADD}] // Reverse

Out[2]= or[member[composite[INTMUL, LEFT[x], PLUS], binhom[NATADD, INTADD]],
          not[member[x, Z]]] == True

In[3]:= (% /. x → x_) /. Equal → SetDelayed
```

Conversely:

```
In[4]:= SubstTest[implies, member[w, binhom[NATADD, INTADD]],
          equal[domain[w], omega], w -> composite[INTMUL, LEFT[x], PLUS]] // Reverse

Out[4]= or[member[x, Z],
          not[member[composite[INTMUL, LEFT[x], PLUS], binhom[NATADD, INTADD]]]] == True

In[5]:= (% /. x → x_) /. Equal → SetDelayed
```

These results can be combined into a single rewrite rule.

```
In[6]:= equiv[member[composite[INTMUL, LEFT[x], PLUS], binhom[NATADD, INTADD]], member[x, Z]]

Out[6]= True

In[7]:= member[composite[INTMUL, LEFT[x_], PLUS], binhom[NATADD, INTADD]] := member[x, Z]
```

The uniqueness theorem for binary homomorphisms from **NATADD** to **INTADD** yields:

```
In[8]:= SubstTest[equal, w, composite[MIXMUL, LEFT[APPLY[w, set[0]]]],
         w -> composite[INTMUL, LEFT[x], PLUS]]

Out[8]= True == equal[composite[MIXMUL, LEFT[x]], composite[INTMUL, LEFT[x], PLUS]]

In[9]:= composite[INTMUL, LEFT[x_], PLUS] := composite[MIXMUL, LEFT[x]]
```

Using **reify**, one can eliminate the variable.

```
In[10]:= Map[rotate[inverse[#]] &, SubstTest[reify, x, composite[z, LEFT[x], PLUS], z → INTMUL]]

Out[10]= composite[INTMUL, cross[Id, PLUS]] == MIXMUL

In[11]:= composite[INTMUL, cross[Id, PLUS]] := MIXMUL
```

Remark. In principle one could use this formula to eliminate **MIXMUL** altogether, but doing so would cut off access to theorems about **MIXMUL**, defeating our purpose.

---

zero divisors

Lemma.

```
In[12]:= Map[composite[PLUS, #] &, IminComp[INTMUL, cross[Id, PLUS], set[id[omega]]]] // Reverse

Out[12]= composite[id[range[PLUS]], image[inverse[INTMUL], set[id[omega]]]] ==
          union[cart[Z, set[id[omega]]], cart[set[id[omega]], range[PLUS]]]

In[13]:= % /. Equal → SetDelayed
```

Lemma.

```
In[14]:= IminComp[INTMUL, cross[Id, INVERSE], set[id[omega]]] // Reverse

Out[14]= composite[INVERSE, image[inverse[INTMUL], set[id[omega]]]] ==
          image[inverse[INTMUL], set[id[omega]]]

In[15]:= composite[INVERSE, image[inverse[INTMUL], set[id[omega]]]] :=
          image[inverse[INTMUL], set[id[omega]]]
```

Lemma.

```
In[16]:= equal[composite[INVERSE, id[range[PLUS]]],
           composite[id[image[INVERSE, range[PLUS]]], INVERSE]] // AssertTest

Out[16]= equal[composite[INVERSE, id[range[PLUS]]],
           composite[id[image[INVERSE, range[PLUS]]], INVERSE]] == True

In[17]:= composite[id[image[INVERSE, range[PLUS]]], INVERSE] :=
          composite[INVERSE, id[range[PLUS]]]
```

Lemma.

```
In[18]:= Assoc[id[image[INVERSE, range[PLUS]]],
          INVERSE, image[inverse[INTMUL], set[id[omega]]]]

Out[18]= composite[id[image[INVERSE, range[PLUS]]], image[inverse[INTMUL], set[id[omega]]]] ==
          union[cart[Z, set[id[omega]]], cart[set[id[omega]], image[INVERSE, range[PLUS]]]]

In[19]:= % /. Equal → SetDelayed
```

Lemma

```
In[20]:= IminComp[INTMUL, id[cart[V, Z]], x] // Reverse

Out[20]= composite[id[Z], image[inverse[INTMUL], x]] == image[inverse[INTMUL], x]

In[21]:= composite[id[Z], image[inverse[INTMUL], x_]] := image[inverse[INTMUL], x]
```

Main theorem.

```
In[22]:= SubstTest[composite, union[u, v], w,
           {u → id[range[PLUS]], v -> id[image[INVERSE, range[PLUS]]],
            w -> image[inverse[INTMUL], set[id[omega]]]}] // Reverse

Out[22]= image[inverse[INTMUL], set[id[omega]]] ==
          union[cart[Z, set[id[omega]]], cart[set[id[omega]], Z]]

In[23]:= image[inverse[INTMUL], set[id[omega]]] :=
          union[cart[Z, set[id[omega]]], cart[set[id[omega]], Z]]
```

multiplying by zero

Theorem. Multiplication by zero is a constant function with value zero.

```
In[24]:= SubstTest[equal, w,
          composite[INTMUL, LEFT[APPLY[w, plus[set[0]]]]], w → cart[Z, set[id[omega]]]]

Out[24]= True == equal[cart[Z, set[id[omega]]], composite[INTMUL, LEFT[id[omega]]]]

In[25]:= composite[INTMUL, LEFT[id[omega]]] := cart[Z, set[id[omega]]]
```

Corollary.

```
In[26]:= Map[A, ImageComp[INTMUL, LEFT[id[omega]], set[x]]] // Reverse

Out[26]= APPLY[INTMUL, PAIR[id[omega], x]] ==
          union[complement[image[V, intersection[Z, set[x]]]], id[omega]]

In[27]:= APPLY[INTMUL, PAIR[id[omega], x_]] :=
          union[complement[image[V, intersection[Z, set[x]]]], id[omega]]
```

consequences of the distributive law

There are two distributive laws, but they are related by the fact that multiplication is commutative. One can use the one to derive the other as follows:

```
In[28]:= Assoc[composite[INTADD, cross[INTMUL, INTMUL], TWIST], cross[Id, DUP], SWAP]

Out[28]= composite[INTADD, cross[INTMUL, INTMUL], TWIST, cross[DUP, Id]] ==
          composite[INTMUL, cross[Id, INTADD]]

In[29]:= composite[INTADD, cross[INTMUL, INTMUL], TWIST, cross[DUP, Id]] :=
          composite[INTMUL, cross[Id, INTADD]]
```

A similar procedure yields another consequence of the distributive law, which basically says that left multiplication by an integer is an endomorphism of **INTADD**. But note that the variable **x** here is not restricted to integers.

```
In[30]:= Assoc[composite[INTADD, cross[INTMUL, INTMUL], TWIST],
           cross[Id, DUP], RIGHT[x]] // Reverse

Out[30]= composite[INTMUL, LEFT[x], INTADD] ==
          composite[INTADD, cross[composite[INTMUL, LEFT[x]], composite[INTMUL, LEFT[x]]]]

In[31]:= composite[INTMUL, LEFT[x_], INTADD] :=
          composite[INTADD, cross[composite[INTMUL, LEFT[x]], composite[INTMUL, LEFT[x]]]]
```

left multiplication by a nonzero integer is one-to-one

Lemma.

```
In[32]:= Assoc[composite[INVERSE, INTMUL], cross[Id, INVERSE], LEFT[x]]

Out[32]= composite[INVERSE, INTMUL, LEFT[x], INVERSE] == composite[INTMUL, LEFT[x]]

In[33]:= composite[INVERSE, INTMUL, LEFT[x_], INVERSE] := composite[INTMUL, LEFT[x]]
```

Lemma.

```
In[34]:= SubstTest[subclass, composite[w, inverse[w]], Id,
           w -> composite[inverse[LEFT[x]], inverse[INTMUL]]] // Reverse

Out[34]= subclass[composite[inverse[LEFT[x]], inverse[INTMUL], INTMUL, LEFT[x]], Id] ==
          FUNCTION[composite[inverse[LEFT[x]], inverse[INTMUL]]]

In[35]:= subclass[composite[inverse[LEFT[x_]], inverse[INTMUL], INTMUL, LEFT[x_]], Id] :=
          FUNCTION[composite[inverse[LEFT[x]], inverse[INTMUL]]]
```

Lemma.

```
In[36]:= member[Z, range[SINGLETON]] // AssertTest

Out[36]= member[Z, range[SINGLETON]] == False

In[37]:= member[Z, range[SINGLETON]] := False
```

Lemma.

```
In[38]:= subclass[composite[id[omega], x, id[omega]], id[omega]] // AssertTest // Reverse

Out[38]= equal[0, intersection[omega, fix[composite[Di, id[omega], inverse[x]]]]] ==
          subclass[composite[id[omega], x, id[omega]], id[omega]]

In[39]:= equal[0, intersection[omega, fix[composite[Di, id[omega], inverse[x_]]]]] :=
          subclass[composite[id[omega], x, id[omega]], id[omega]]
```

Lemma.

```
In[40]:= Map[not, SubstTest[and, subclass[x, y], subclass[intersection[x, y], z],
            {y -> cart[omega, omega], z -> id[omega]}]] // Reverse

Out[40]= or[not[subclass[x, cart[omega, omega]]],
           not[subclass[composite[id[omega], x, id[omega]], id[omega]]]] ==
          not[subclass[x, id[omega]]]

In[41]:= or[not[subclass[x_, cart[omega, omega]]],
           not[subclass[composite[id[omega], x_, id[omega]], id[omega]]]] :=
          not[subclass[x, id[omega]]]
```

Lemma.

```
In[42]:= equiv[or[not[member[x, V]], not[subclass[omega, fix[x]]], not[subclass[x, id[omega]]]],
           not[equal[x, id[omega]]]] // assert

Out[42]= True

In[43]:= or[not[member[x_, V]], not[subclass[omega, fix[x_]]], not[subclass[x_, id[omega]]]] :=
          not[equal[x, id[omega]]]
```

Theorem. Left multiplication by **x** is one-to-one if and only if **x** is not the integer zero (= **id[omega]**).

```
In[44]:= Map[subclass[inverse[#], Id] &,
          IminComp[composite[INTMUL, LEFT[x]], rotate[INTADD], set[id[omega]]]]

Out[44]= FUNCTION[composite[inverse[LEFT[x]], inverse[INTMUL]]] == not[equal[x, id[omega]]]

In[45]:= FUNCTION[composite[inverse[LEFT[x_]], inverse[INTMUL]]] := not[equal[x, id[omega]]]
```

---

eliminating the variable

Lemma.

```
In[46]:= (member[x, image[inverse[funpart[w]], y]] // AssertTest) /. w → INTTIMES

Out[46]= member[x, image[inverse[INTTIMES], y]] ==
          and[member[x, Z], member[composite[INTMUL, LEFT[x]], y]]

In[47]:= member[x_, image[inverse[INTTIMES], y_]] :=
          and[member[x, Z], member[composite[INTMUL, LEFT[x]], y]]
```

Theorem.

```
In[48]:= image[inverse[INTTIMES], BIJ] // Normality

Out[48]= image[inverse[INTTIMES], BIJ] == intersection[Z, complement[set[id[omega]]]]

In[49]:= image[inverse[INTTIMES], BIJ] := intersection[Z, complement[set[id[omega]]]]
```

Comment. The analog in natural arithmetic is this:

```
In[50]:= image[inverse[TIMES], BIJ] // Normality

Out[50]= image[inverse[TIMES], BIJ] == intersection[omega, complement[set[0]]]

In[51]:= image[inverse[TIMES], BIJ] := intersection[omega, complement[set[0]]]
```

division of integers

Division of integers corresponds to a binary function obtained by rotating **INTMUL**, and restricting denominators to be nonzero. The main tool is the rule that left-multiplication by a nonzero integer is one-to-one. The trick to cleaning up expressions that are encountered is the following observation:

```
In[52]:= implies[equal[0, fix[composite[u, v]]], equal[0, fix[composite[v, u]]]]

Out[52]= True
```

The stratgy will be to rewrite this fact:

```
In[53]:= FUNCTION[composite[inverse[LEFT[x]], inverse[INTMUL]]]

Out[53]= not[equal[x, id[omega]]]
```

The following class is needed several times.

```
In[54]:= class[w, not[equal[x, id[omega]]]]

Out[54]= union[image[V, intersection[Di, x]], image[V, intersection[omega, complement[fix[x]]]],
          image[V, intersection[x, complement[cart[omega, V]]]]]
```

For clarity, a temporary abbreviation for it will be used:

```
In[55]:= temp[x_] := union[image[V, intersection[Di, x]],
           image[V, intersection[omega, complement[fix[x]]]],
           image[V, intersection[x, complement[cart[omega, V]]]]]
```

The first step is to rewrite the **FUNCTION** statement as follows:

```
In[56]:= SubstTest[FUNCTION,
           composite[id[image[V, t]], inverse[LEFT[x]], inverse[INTMUL]], t → temp[x]] // Reverse

Out[56]= FUNCTION[
           composite[id[union[image[V, intersection[Di, x]], image[V, intersection[omega,
                complement[fix[x]]]], image[V, intersection[x, complement[cart[omega, V]]]]]],
            inverse[LEFT[x]], inverse[INTMUL]]] == True

In[57]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[58]:= SubstTest[and, subclass[x, y], subclass[y, x], y → id[omega]] // Reverse

Out[58]= and[subclass[omega, fix[x]], subclass[x, id[omega]]] == equal[x, id[omega]]

In[59]:= and[subclass[omega, fix[x_]], subclass[x_, id[omega]]] := equal[x, id[omega]]
```

Eliminating the variable **x** is accomplished as follows:

```
In[60]:= SubstTest[assert,
           forall[x, FUNCTION[composite[id[image[V, temp[x]]], inverse[LEFT[x]], inverse[v]]]],
           v → INTMUL] // InvertFix

Out[60]= subclass[fix[composite[FIRST, intersection[composite[inverse[INTMUL], INTMUL],
             composite[inverse[SECOND], Di, FIRST]], inverse[SECOND]]], set[id[omega]]] == True

In[61]:= % /. Equal → SetDelayed
```

This result can be reformulated as follows:

```
In[62]:= SubstTest[implies, equal[0, fix[composite[u, v]]], equal[0, fix[composite[v, u]]],
             {u -> composite[FIRST, intersection[
               composite[inverse[INTMUL], FIRST], composite[inverse[SECOND], Di, SECOND]]],
                 v -> composite[inverse[rotate[INTMUL]], id[complement[set[id[omega]]]]]}] //
           Reverse

Out[62]= equal[0, fix[composite[SWAP,
             cross[id[complement[set[id[omega]]]], Di], inverse[INTMUL], INTMUL]]] == True

In[63]:= fix[composite[SWAP,
            cross[id[complement[set[id[omega]]]], Di], inverse[INTMUL], INTMUL]] := 0
```

A second application of this idea removes the **SWAP**.

```
In[64]:= SubstTest[implies, equal[0, fix[composite[u, v]]], equal[0, fix[composite[v, u]]],
             {u -> composite[SWAP, cross[id[complement[set[id[omega]]]], Di], inverse[INTMUL]],
           v → INTMUL}]

Out[64]= True == equal[0,
           fix[composite[INTMUL, cross[id[complement[set[id[omega]]]], Di], inverse[INTMUL]]]]

In[65]:= fix[composite[INTMUL, cross[id[complement[set[id[omega]]]], Di], inverse[INTMUL]]] := 0
```

The final step is to rewrite this in a more recognizable form:

```
In[66]:= SubstTest[equal, 0, fix[composite[w, cross[Id, Di], inverse[w]]],
          w -> composite[INTMUL, cross[id[complement[set[id[omega]]]], Id]]]

Out[66]= FUNCTION[composite[rotate[INTMUL], id[cart[V, complement[set[id[omega]]]]]]] == True

In[67]:= FUNCTION[composite[rotate[INTMUL], id[cart[V, complement[set[id[omega]]]]]]] := True
```