INTMUL, part 2. commutative law

Johan G. F. Belinfante
2006 December 25

```
In[1]:= SetDirectory["l:"]; << goedel88.24a; << tools.m

        :Package Title: goedel88.24a        2006 December 24 at 5:40 p.m.

        It is now:  2006 Dec 25 at 17:3

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 December 17

        weightlimit = 40
```

summary

The commutative law of integer multiplication is derived in this notebook. One of the principal ideas is to make use of the uniqueness theorem for endomorphisms of **INTADD**: Every endomorphism of integer addition is of the form **t = composite[-INTMUL, LEFT[v]]**, where **v** is the value of **t** at the integer **+1**. To establish the fact that **INTMUL** is commutative, one needs to show that left multiplication and right multiplication by any integer coincide. To do that the following observation is central: right multiplication by an integer **x** is the composite of the function **INTTIMES** and evaluation at **x**.

```
In[2]:= composite[eval[x], INTTIMES]

Out[2]= composite[INTMUL, RIGHT[x]]
```

The function **INTTIMES** is a binary homomorphism from integer addition to the addition of endomorphisms. The restriction of evaluation to endomorphsms is a binary homomorphism in the opposite direction, and therefore their composite is an endomorphism of integer addition.

evaluating pointwise sums of functions

For any binary operation **x** there is a pointwise binary operation **composite[IMAGE[cross[inverse[DUP], funpart[x]]], CROSS]** for functions. The process of evaluation is a binary homomorphism from this pointwise operation to the operation **x**. This general fact is derived as follows:

```
In[3]:= Map[VERTSECT, SubstTest[reify, t, APPLY[funpart[z],
          composite[funpart[x], intersection[composite[inverse[FIRST], funpart[first[t]]],
            composite[inverse[SECOND], funpart[second[t]]]]], z → eval[w]]]

Out[3]= composite[eval[w], IMAGE[cross[inverse[DUP], funpart[x]]], CROSS,
          cross[FUNPART, FUNPART]] == composite[funpart[x], cross[eval[w], eval[w]]]
```

```
In[4]:= composite[eval[w_], IMAGE[cross[inverse[DUP], funpart[x_]]], CROSS,
          cross[FUNPART, FUNPART]] := composite[funpart[x], cross[eval[w], eval[w]]]
```

Specializing to the case of **INTADD**, one finds:

```
In[5]:= SubstTest[composite, eval[x], IMAGE[cross[inverse[DUP], funpart[y]]],
          CROSS, cross[FUNPART, FUNPART], y → INTADD] // Reverse

Out[5]= composite[eval[x], IMAGE[cross[inverse[DUP], INTADD]], CROSS, cross[FUNPART, FUNPART]] ==
         composite[INTADD, cross[eval[x], eval[x]]]
```

```
In[6]:= composite[eval[x_], IMAGE[cross[inverse[DUP], INTADD]], CROSS,
          cross[FUNPART, FUNPART]] := composite[INTADD, cross[eval[x], eval[x]]]
```

Corollary.

```
In[7]:= Assoc[eval[x], composite[IMAGE[cross[inverse[DUP], INTADD]],
          CROSS, cross[FUNPART, FUNPART]], id[cartsq[binhom[INTADD, INTADD]]]]

Out[7]= composite[eval[x], IMAGE[cross[inverse[DUP], INTADD]],
          CROSS, id[cart[binhom[INTADD, INTADD], binhom[INTADD, INTADD]]]] ==
         composite[INTADD, cross[composite[eval[x], id[binhom[INTADD, INTADD]]],
          composite[eval[x], id[binhom[INTADD, INTADD]]]]]
```

```
In[8]:= composite[eval[x_], IMAGE[cross[inverse[DUP], INTADD]],
          CROSS, id[cart[binhom[INTADD, INTADD], binhom[INTADD, INTADD]]]] :=
         composite[INTADD, cross[composite[eval[x], id[binhom[INTADD, INTADD]]],
          composite[eval[x], id[binhom[INTADD, INTADD]]]]]
```

rewrite rules for two logical equivalences

Logical equivalences can be made into rewrite rules. For example, the statement that **w** is a mapping from **u** to **v** can be stated as an equivalence and made to a rewrite rule:

```
In[9]:= equiv[and[member[w, FUNS], equal[domain[w], u], subclass[range[w], v]],
          member[w, map[u, v]]] // not // not

Out[9]= True
```

```
In[10]:= and[equal[u_, domain[w_]], FUNCTION[w_],
           member[w_, V], subclass[range[w_], v_]] := member[w, map[u, v]]
```

An application:

```
In[13]:= SubstTest[and, member[w, FUNS], equal[domain[w], u],
           subclass[range[w], v], {u -> binhom[INTADD, INTADD],
           v -> Z, w -> composite[eval[x], id[binhom[INTADD, INTADD]]]}]

Out[13]= member[composite[eval[x], id[binhom[INTADD, INTADD]]],
           map[binhom[INTADD, INTADD], Z]] == member[x, Z]
```

```
In[14]:= member[composite[eval[x_], id[binhom[INTADD, INTADD]]],
           map[binhom[INTADD, INTADD], Z]] := member[x, Z]
```

Likewise, the statement that **w** is a binary homomorphism from **u** to **v** can be stated as a logical equivalence and made into a rewrite rule.

```
In[11]:= equiv[and[member[w, map[fix[domain[u]], fix[domain[v]]]], equal[composite[w, u],
             composite[v, cross[w, w]]]], member[w, binhom[u, v]]] // not // not

Out[11]= True
```

```
In[12]:= and[equal[composite[v_, cross[w_, w_]], composite[w_, u_]],
           member[w_, map[fix[domain[u_]], fix[domain[v_]]]]] := member[w, binhom[u, v]]
```

Application:

```
In[15]:= SubstTest[and, member[w, map[fix[domain[u]], fix[domain[v]]]],
          equal[composite[w, u], composite[v, cross[w, w]]],
          {u → composite[IMAGE[cross[inverse[DUP], INTADD]],
             CROSS, id[cartsq[binhom[INTADD, INTADD]]]],
           v → INTADD, w → composite[eval[x], id[binhom[INTADD, INTADD]]]}]

Out[15]= member[composite[eval[x], id[binhom[INTADD, INTADD]]],
           binhom[composite[IMAGE[cross[inverse[DUP], INTADD]], CROSS,
             id[cart[binhom[INTADD, INTADD], binhom[INTADD, INTADD]]]], INTADD]] == member[x, Z]
```

```
In[16]:= member[composite[eval[x_], id[binhom[INTADD, INTADD]]],
           binhom[composite[IMAGE[cross[inverse[DUP], INTADD]], CROSS,
             id[cart[binhom[INTADD, INTADD], binhom[INTADD, INTADD]]]], INTADD]] := member[x, Z]
```

---

composite of eval and INTTIMES

The composite of binary homomorphisms is a binary homomorphism. In particular, this can be applied to the case of the restriction of **eval[x]** to endomorphisms of integer addition and the binary homomorphism **INTTIMES** which assigns to each integer the endomorphism of multiplying on the left by that integer:

```
In[17]:= SubstTest[implies, and[member[s, binhom[v, w]], member[t, binhom[u, v]]],
           member[composite[s, t], binhom[u, w]],
           {s -> composite[eval[x], id[binhom[INTADD, INTADD]]], t → INTTIMES,
            u → INTADD, v -> composite[IMAGE[cross[inverse[DUP], INTADD]], CROSS,
              id[cart[binhom[INTADD, INTADD], binhom[INTADD, INTADD]]]], w -> INTADD}] // Reverse

Out[17]= or[member[composite[INTMUL, RIGHT[x]], binhom[INTADD, INTADD]], not[member[x, Z]]] ==
          True
```

```
In[18]:= (% /. x → x_) /. Equal → SetDelayed
```

Conversely:

```
In[19]:= SubstTest[implies, member[w, binhom[INTADD, INTADD]],
           equal[domain[w], Z], w -> composite[INTMUL, RIGHT[x]]] // Reverse

Out[19]= or[member[x, Z],
           not[member[composite[INTMUL, RIGHT[x]], binhom[INTADD, INTADD]]]] == True

In[20]:= (% /. x → x_) /. Equal → SetDelayed
```

One can express this as a logical equivalence, and introduce a corresponding rewrite rule:

```
In[21]:= equiv[member[composite[INTMUL, RIGHT[x]], binhom[INTADD, INTADD]], member[x, Z]]

Out[21]= True

In[22]:= member[composite[INTMUL, RIGHT[x_]], binhom[INTADD, INTADD]] := member[x, Z]
```

left multiplication by +1

The identity function i**d[Z]** on the set **Z** of integers preserves addition, and therefore by the uniqueness theorem for endomorphisms, it is a left nultiplication. Its value at **+1** is obviously **+1**, so it can be identified as the operation of left multiplication by **+1**.

```
In[23]:= SubstTest[equal, x, composite[INTMUL, LEFT[APPLY[x, plus[set[0]]]]], x → id[Z]]

Out[23]= True == equal[composite[INTMUL, LEFT[composite[id[omega], SUCC]]], id[Z]]

In[24]:= composite[INTMUL, LEFT[composite[id[omega], SUCC]]] := id[Z]
```

Corollary. The product of **+1** and an integer **x** is **x**. If **x** is not an integer, the application yields the universal class **V**. Both cases are covered by the following rewite rule:

```
In[25]:= Map[A, ImageComp[INTMUL, LEFT[composite[id[omega], SUCC]], set[x]]] // Reverse

Out[25]= APPLY[INTMUL, PAIR[composite[id[omega], SUCC], x]] ==
          union[x, complement[image[V, intersection[Z, set[x]]]]]

In[26]:= APPLY[INTMUL, PAIR[composite[id[omega], SUCC], x_]] :=
          union[x, complement[image[V, intersection[Z, set[x]]]]]
```

the commutative law

Lemma. (A simplification rule.)

```
In[27]:= equal[composite[INTMUL, LEFT[x], id[image[V, intersection[Z, set[x]]]]],
          composite[INTMUL, LEFT[x]]]

Out[27]= True
```

```
In[28]:= composite[INTMUL, LEFT[x_], id[image[V, intersection[Z, set[x_]]]]] :=
          composite[INTMUL, LEFT[x]]
```

Theorem. Applying the uniqueness theorem to right multiplication, one finds that right multiplication by **x** is the same as left multiplication by **x**.

```
In[29]:= SubstTest[equal, w, composite[INTMUL, LEFT[APPLY[w, plus[set[0]]]]],
          w -> composite[INTMUL, RIGHT[x]]]

Out[29]= True == equal[composite[INTMUL, LEFT[x]], composite[INTMUL, RIGHT[x]]]

In[30]:= composite[INTMUL, RIGHT[x_]] := composite[INTMUL, LEFT[x]]
```

The variable **x** can be eliminated by reification.

```
In[31]:= Map[rotate[inverse[#]] &, SubstTest[reify, x, composite[INTMUL, f[x]], f → RIGHT]]

Out[31]= composite[INTMUL, SWAP] == INTMUL

In[32]:= composite[INTMUL, SWAP] := INTMUL
```