binary homomorphisms, part 4

Johan G. F. Belinfante
2006 July 23

```
In[1]:= SetDirectory["l:"]; << goedel83.20a; << tools.m

        :Package Title: goedel83.20a         2006 July 20 at 5:00 p.m.

        It is now:  2006 Jul 23 at 9:44

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 July 18

        weightlimit = 40
```

summary

This is the fourth in a series of notebooks about homomorphisms of binary operations. A set **x** is said to be an **idempotent** of a binary operation **y** if **member[pair[pair[x, x], x], y]**. An **endomorphism** of a binary operation **y** is a binary homomorphism from **y** to **y**. In this notebook, the following theorem is derived, and illustrated with examples: If **x** is an idempotent for a binary operation **y**, then the constant function **cart[fix[domain[y]], set[x]]** is an endomorphism of **y**.

derivation

Lemma 1.

```
In[2]:= or[member[pair[pair[x, x], x], y], not[member[x, fix[composite[y, DUP]]]]] // AssertTest

Out[2]= or[member[pair[pair[x, x], x], y], not[member[x, fix[composite[y, DUP]]]]] == True

In[3]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma 2.

```
In[4]:= or[equal[x, APPLY[y, PAIR[x, x]]], not[FUNCTION[y]],
          not[member[x, fix[composite[y, DUP]]]]] // AssertTest

Out[4]= or[equal[x, APPLY[y, PAIR[x, x]]],
          not[FUNCTION[y]], not[member[x, fix[composite[y, DUP]]]]] == True

In[5]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Corollary.

```
In[6]:= Map[not, SubstTest[and, implies[p1, p3],
          implies[and[p2, p3], p4], not[implies[and[p1, p2], p4]],
          {p1 -> member[y, BINOPS], p2 -> member[x, fix[composite[y, DUP]]],
           p3 → FUNCTION[y], p4 -> equal[x, APPLY[y, PAIR[x, x]]]}]]

Out[6]= or[equal[x, APPLY[y, PAIR[x, x]]],
          not[member[x, fix[composite[y, DUP]]]], not[member[y, BINOPS]]] == True

In[7]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma 3.

```
In[8]:= SubstTest[implies, and[equal[x, APPLY[y, PAIR[x, x]]], equal[y, funpart[z]]],
         equal[image[y, cart[set[x], set[x]]], set[x]], z → y]

Out[8]= or[equal[image[y, cart[set[x], set[x]]], set[x]],
          not[equal[x, APPLY[y, PAIR[x, x]]]], not[FUNCTION[y]]] == True

In[9]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Corollary.

```
In[10]:= Map[not, SubstTest[and, implies[p1, p3], implies[and[p2, p3], p4],
           implies[and[p1, p2], p4], implies[and[p3, p4], p5], not[implies[and[p1, p2], p5]],
           {p1 -> member[y, BINOPS], p2 -> member[x, fix[composite[y, DUP]]],
            p3 → FUNCTION[y], p4 -> equal[x, APPLY[y, PAIR[x, x]]],
            p5 → equal[set[x], image[y, cart[set[x], set[x]]]]}]]

Out[10]= or[equal[image[y, cart[set[x], set[x]]], set[x]],
           not[member[x, fix[composite[y, DUP]]]], not[member[y, BINOPS]]] == True

In[11]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma 4.

```
In[12]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[p1, p4],
           implies[p4, p5], implies[and[p1, p2], p6], implies[and[p0, p1, p3, p5, p6], p7],
           not[implies[and[p0, p1, p2], p7]], {p0 → equal[w, cart[fix[domain[y]], set[x]]],
            p1 → member[y, BINOPS], p2 → member[x, fix[composite[y, DUP]]],
            p3 → equal[x, APPLY[y, PAIR[x, x]]], p4 → subclass[range[y], fix[domain[y]]],
            p5 -> equal[domain[y], image[inverse[y], fix[domain[y]]]],
            p6 -> equal[image[y, cart[set[x], set[x]]], set[x]],
            p7 → equal[composite[w, y], composite[y, cross[w, w]]]}]]

Out[12]= or[equal[composite[w, y], composite[y, cross[w, w]]],
           not[equal[w, cart[fix[domain[y]], set[x]]]],
           not[member[x, fix[composite[y, DUP]]]], not[member[y, BINOPS]]] == True

In[13]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma 5.

```
In[14]:= Map[not, SubstTest[and, implies[p1, p5],
           implies[p1, p3], implies[and[p1, p3], p6], implies[and[p5, p6], p7],
           implies[p1, p8], implies[and[p1, p7, p8], p9], not[implies[p1, p9]],
           {p1 → and[equal[w, cart[fix[domain[y]], set[x]]], member[y, BINOPS],
              member[x, fix[composite[y, DUP]]]], p3 → equal[x, APPLY[y, PAIR[x, x]]],
            p5 → subclass[range[y], fix[domain[y]]], p6 → member[x, range[y]],
            p7 → member[x, fix[domain[y]]], p8 → member[fix[domain[y]], V],
            p9 → member[w, map[fix[domain[y]], fix[domain[y]]]]}]]

Out[14]= or[member[w, map[fix[domain[y]], fix[domain[y]]]],
           not[equal[w, cart[fix[domain[y]], set[x]]]],
           not[member[x, fix[composite[y, DUP]]]], not[member[y, BINOPS]]] == True

In[15]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Main theorem. There is a constant endomorphism corresponding to any idempotent of a binary operation.

```
In[16]:= Map[not, SubstTest[and, implies[p1, p4],
            implies[p1, p3], implies[and[p3, p4], p5], not[implies[p1, p5]],
            {p1 → and[equal[w, cart[fix[domain[y]], set[x]]], member[y, BINOPS], member[x,
                fix[composite[y, DUP]]]], p3 → member[w, map[fix[domain[y]], fix[domain[y]]]],
             p4 → equal[composite[w, y], composite[y, cross[w, w]]],
             p5 → member[w, binhom[y, y]]}]] /. w -> cart[fix[domain[y]], set[x]]

Out[16]= or[member[cart[fix[domain[y]], set[x]], binhom[y, y]],
           not[member[x, fix[composite[y, DUP]]]], not[member[y, BINOPS]]] == True

In[17]:= or[member[cart[fix[domain[y_]], set[x_]], binhom[y_, y_]],
           not[member[x_, fix[composite[y_, DUP]]]], not[member[y_, BINOPS]]] := True
```

membership rule for the class of idempotents

A general membership rule can be derived for the class of all idempotents of any binary operation. This general rule is added here only on a temporary basis because, as will be seen below, it does cause some problems in practice.

```
In[18]:= member[x, fix[composite[y, DUP]]] // AssertTest

Out[18]= member[x, fix[composite[y, DUP]]] == and[member[x, V], member[pair[pair[x, x], x], y]]

In[19]:= member[x_, fix[composite[y_, DUP]]] := and[member[x, V], member[pair[pair[x, x], x], y]]
```

The membership rule for the class of all idempotents derived in this section is used in the remaining sections to find all idempotents of **NATADD**, **NATMUL** and **INTADD**.

NATADD (addition of natural numbers)

A technical difficulty is that the **GOEDEL** program has an explicit membership rule for **NATADD** which generally yields expressions involving **iterate**. The key observation is that one can avoid **iterate** by using a **nat** wrapper. The following result is then obtained, and all that remains to be done is just to eliminate the variable **x**.

```
In[20]:= member[nat[x], fix[composite[NATADD, DUP]]]
Out[20]= equal[0, nat[x]]
```

Lemma.

```
In[21]:= SubstTest[fix, composite[id[omega], x, DUP], x → NATADD] // Reverse
Out[21]= intersection[omega, fix[composite[NATADD, DUP]]] == fix[composite[NATADD, DUP]]

In[22]:= % /. Equal → SetDelayed
```

Eliminating the variable **x** yields an inclusion:

```
In[23]:= Map[equal[V, #] &, SubstTest[class, x, implies[member[nat[x], y], equal[0, nat[x]]],
           y -> fix[composite[NATADD, DUP]]]] // Reverse
Out[23]= subclass[fix[composite[NATADD, DUP]], set[0]] == True

In[24]:= % /. Equal → SetDelayed
```

At this point, the **GOEDEL** program is automatically able to recognize that the following equation holds:

```
In[25]:= equal[fix[composite[NATADD, DUP]], set[0]]
Out[25]= True

In[26]:= fix[composite[NATADD, DUP]] := set[0]
```

Corresponding to the single idempotent **0**, there is a constant endomorphism of **NATADD**, usually called the zero endomorphism:

```
In[27]:= member[cart[omega, set[0]], binhom[NATADD, NATADD]]
Out[27]= True
```

NATMUL (multiplication of natural numbers)

The membership rule for **NATMUL** does not automatically open up, but it does so when **AssertTest** is applied. However, the result again simplifies when a **nat** wrapper is introduced:

```
In[28]:= (member[pair[pair[y, y], y], NATMUL] // AssertTest) /. y → nat[x]

Out[28]= member[pair[pair[nat[x], nat[x]], nat[x]], NATMUL] ==
          or[equal[0, nat[x]], equal[nat[x], set[0]]]

In[29]:= member[pair[pair[nat[x_], nat[x_]], nat[x_]], NATMUL] :=
          or[equal[0, nat[x]], equal[nat[x], set[0]]]
```

Removing the **nat** wrapper, one finds:

```
In[30]:= SubstTest[implies, and[equal[x, nat[y]], member[x, fix[composite[NATMUL, DUP]]]],
          or[equal[0, x], equal[set[0], x]], y → x]

Out[30]= or[equal[0, x], equal[x, set[0]],
           not[member[x, omega]], not[member[pair[pair[x, x], x], NATMUL]]] == True

In[31]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[32]:= SubstTest[fix, composite[id[omega], x, DUP], x → NATMUL] // Reverse

Out[32]= intersection[omega, fix[composite[NATMUL, DUP]]] == fix[composite[NATMUL, DUP]]

In[33]:= % /. Equal → SetDelayed
```

Eliminating the variable **x** yields an inclusion:

```
In[34]:= Map[equal[V, #] &, SubstTest[class, x, or[equal[0, x], equal[x, set[0]],
             not[member[x, omega]], not[member[pair[pair[x, x], x], y]]], y → NATMUL]] // Reverse

Out[34]= subclass[fix[composite[NATMUL, DUP]], succ[set[0]]] == True

In[35]:= % /. Equal → SetDelayed
```

To obtain an inclusion in the reverse direction, one needs these facts:

```
In[36]:= member[pair[pair[0, 0], 0], NATMUL] // AssertTest

Out[36]= member[pair[pair[0, 0], 0], NATMUL] == True

In[37]:= member[pair[pair[0, 0], 0], NATMUL] := True

In[38]:= member[pair[pair[set[0], set[0]], set[0]], NATMUL] // AssertTest

Out[38]= member[pair[pair[set[0], set[0]], set[0]], NATMUL] == True

In[39]:= member[pair[pair[set[0], set[0]], set[0]], NATMUL] := True
```

Combining these results yields an equation:

```
In[40]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> fix[composite[NATMUL, DUP]], v -> succ[set[0]]}]

Out[40]= True == equal[fix[composite[NATMUL, DUP]], succ[set[0]]]
```

```
In[41]:= fix[composite[NATMUL, DUP]] := succ[set[0]]
```

For each of the two idempotents **0** and **set[0]**, there is a corresponding constant endomorphism of **NATMUL**.

```
In[42]:= member[cart[omega, set[0]], binhom[NATMUL, NATMUL]] // AssertTest
Out[42]= member[cart[omega, set[0]], binhom[NATMUL, NATMUL]] == True

In[43]:= member[cart[omega, set[0]], binhom[NATMUL, NATMUL]] := True
```

For the other case, no new rewrite rule is needed.

```
In[44]:= member[cart[omega, set[set[0]]], binhom[NATMUL, NATMUL]]
Out[44]= True
```

INTADD (integer addition)

As for the case of **NATADD**, a technical difficulty is again encountered due to the presence of an explicit membership rule for **INTADD**. In the case of **INTADD**, it turned out that this did not really matter much because the **GOEDEL** program just happened to contain a rewrite rule for the intersection of the class **Z** of integers and the class **TRV** of transitive relations. The general membership rule introduced in the first section yields this (perhaps somewhat surprising) result:

```
In[45]:= member[x, fix[composite[INTADD, DUP]]]
Out[45]= and[member[x, Z], TRANSITIVE[composite[Id, x]]]
```

Lemma.

```
In[46]:= AssInt[Z, P[cart[V, V]], image[inverse[IMAGE[id[cart[V, V]]]], TRV]] // Reverse
Out[46]= intersection[Z, image[inverse[IMAGE[id[cart[V, V]]]], TRV]] == set[id[omega]]

In[47]:= intersection[Z, image[inverse[IMAGE[id[cart[V, V]]]], TRV]] := set[id[omega]]
```

Theorem. The only idempotent of integer addition is the integer zero. (Recall that the integer zero is **id[omega]**.)

```
In[48]:= fix[composite[INTADD, DUP]] // Normality
Out[48]= fix[composite[INTADD, DUP]] == set[id[omega]]

In[49]:= fix[composite[INTADD, DUP]] := set[id[omega]]
```

Corresponding to this single idempotent, there is a constant endomorphism of **INTADD**, called the zero endomorphism for **INTADD**.

```
In[50]:= member[cart[Z, set[id[omega]]], binhom[INTADD, INTADD]] // AssertTest
Out[50]= member[cart[Z, set[id[omega]]], binhom[INTADD, INTADD]] == True

In[51]:= member[cart[Z, set[id[omega]]], binhom[INTADD, INTADD]] := True
```

comment: some trivial binary homomorphisms

The empty function is a binary homomorphism only when **x** is trivial:

```
In[52]:= member[0, binhom[x, y]] // AssertTest
Out[52]= member[0, binhom[x, y]] == equal[0, fix[domain[x]]]

In[53]:= member[0, binhom[x_, y_]] := equal[0, fix[domain[x]]]
```

The only binary homomorphism from the empty binary operation is the empty homomorphism.

```
In[54]:= binhom[0, x] // Normality
Out[54]= binhom[0, x] == set[0]

In[55]:= binhom[0, x_] := set[0]
```

There are no homomorphisms from **x** to **0** unless **x = 0**.

```
In[56]:= binhom[x, 0] // Normality
Out[56]= binhom[x, 0] == intersection[complement[image[V, fix[domain[x]]]], set[0]]

In[57]:= binhom[x_, 0] := intersection[complement[image[V, fix[domain[x]]]], set[0]]
```

Thus, for the binary operation **0**, there are no idempotents, but there is nonetheless a constant endomorphism, namely **0**.