binary homomorphisms, part 1.

Johan G. F. Belinfante
2006 July 18

```
In[1]:= SetDirectory["l:"]; << goedel83.14a; << tools.m

        :Package Title: goedel83.14a         2006 July 14 at 9:45 p.m.

        It is now:  2006 Jul 18 at 10:32

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 July 13

        weightlimit = 40
```

summary

This is the first in a series of notebooks about homomorphisms of binary systems. The definition of binary homomorphism has been introduced via a membership rule wrapped with **class** to prevent the definition from being expanded out each time this concept is used.

```
In[2]:= Begin["Goedel`Private`"];

In[3]:= FirstMatch[class[w_, member[z_, HoldPattern[binhom[x_, y_]]]]]

Out[3]= class[w_, member[z_, binhom[x_, y_]]] := class[w, and[member[z, map[fix[domain[
          x]], fix[domain[y]]]], equal[composite[z, x], composite[y, cross[z, z]]]]]
```

In this notebook, an equation for **binhom[x, y]** is derived, and rewrite rules characterizing binary homomorphisms are derived to facilitate reasoning about binary homomorphisms. As a simple application, it is shown that the composite of binary homomorphisms is a binary homomorphism. A **reify** rule is obtained from the equation for **binhom[x, y]**. In later notebooks of this series it is planned to provide examples and applications of binary homomorphisms to arithmetic and abstract algebra.

a membership rule

To get started, the **simplify** flag is cleared to speed things up. Later it can be set.

```
In[4]:= simplify = False;
```

A general membership rule for the class of **pair[u,v]** such that **equal[image[x,v], image[y,u]]** is derived using **AssertTest**.

```
In[5]:= (member[pair[u, v], composite[inverse[funpart[z]], w]] // AssertTest) /.
         {z → IMAGE[x], w → IMAGE[y]}

Out[5]= member[pair[u, v], composite[inverse[IMAGE[x]], IMAGE[y]]] ==
         and[equal[image[x, v], image[y, u]], member[u, V], member[v, V], member[image[x, v], V]]

In[6]:= member[pair[u_, v_], composite[inverse[IMAGE[x_]], IMAGE[y_]]] :=
         and[equal[image[x, v], image[y, u]], member[u, V], member[v, V], member[image[x, v], V]]
```

The following application of this membership rule will be needed to derive an equation for **binhom[x, y]**.

```
In[7]:= Map[implies[#, equal[composite[w, x], composite[y, cross[w, w]]]] &,
         (member[w, fix[z]] // AssertTest) /.
          z -> composite[inverse[IMAGE[cross[inverse[x], Id]]], IMAGE[cross[Id, y]], CROSS, DUP]]

Out[7]= or[equal[composite[w, x], composite[y, cross[w, w]]],
          not[member[w, fix[composite[inverse[IMAGE[cross[inverse[x], Id]]],
              IMAGE[cross[Id, y]], CROSS, DUP]]]]] == True

In[8]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

To obtain a result in the other direction, the following lemma is needed:

```
In[9]:= Map[not, SubstTest[and, implies[and[p1, p2], p4],
           implies[and[p1, p3, p4], p5], implies[and[p2, p4, p5], p6],
           not[implies[and[p1, p2, p3], p6]], {p1 → member[w, V], p2 -> equal[z, composite[w, x]],
            p3 -> equal[z, composite[y, cross[w, w]]], p4 → member[range[z], V],
            p5 → member[domain[z], V], p6 → member[z, V]}]] /. z → composite[w, x]

Out[9]= or[member[composite[w, x], V],
          not[equal[composite[w, x], composite[y, cross[w, w]]]], not[member[w, V]]] == True

In[10]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

With this lemma in place, one finds:

```
In[11]:= Map[implies[and[member[w, V], equal[composite[w, x], composite[y, cross[w, w]]]], #] &,
           (member[w, fix[z]] // AssertTest) /.
            z -> composite[inverse[IMAGE[cross[inverse[x], Id]]],
              IMAGE[cross[Id, y]], CROSS, DUP]] // MapNotNot

Out[11]= or[member[w, fix[composite[
              inverse[IMAGE[cross[inverse[x], Id]]], IMAGE[cross[Id, y]], CROSS, DUP]]],
            not[equal[composite[w, x], composite[y, cross[w, w]]]], not[member[w, V]]] == True

In[12]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Combining the two implications yields a logical equivalence that is made into a rewrite rule.

```
In[13]:= equiv[member[w, fix[
             composite[inverse[IMAGE[cross[inverse[x], Id]]], IMAGE[cross[Id, y]], CROSS, DUP]]],
           and[equal[composite[w, x], composite[y, cross[w, w]]], member[w, V]]]

Out[13]= True
```

```
In[14]:= member[w_, fix[composite[
             inverse[IMAGE[cross[inverse[x_], Id]]], IMAGE[cross[Id, y_]], CROSS, DUP]]] :=
           and[equal[composite[w, x], composite[y, cross[w, w]]], member[w, V]]
```

an equation for binhom[x, y]

A standard argument using **symdif** and **Normality** yields the following explicit equation for **binhom[x, y]**. (Comment. This takes quite a while, even with the **simplify** flag cleared.)

```
In[15]:= Map[equal[0, #] &,
           symdif[binhom[x, y], intersection[map[fix[domain[x]], fix[domain[y]]],
             fix[composite[inverse[IMAGE[cross[inverse[x], Id]]],
               IMAGE[cross[Id, y]], CROSS, DUP]]]] // Normality]

Out[15]= equal[binhom[x, y], intersection[fix[
             composite[inverse[IMAGE[cross[inverse[x], Id]]], IMAGE[cross[Id, y]], CROSS, DUP]],
            map[fix[domain[x]], fix[domain[y]]]]] == True

In[16]:= intersection[fix[composite[
             inverse[IMAGE[cross[inverse[x_], Id]]], IMAGE[cross[Id, y_]], CROSS, DUP]],
           map[fix[domain[x_]], fix[domain[y_]]]] := binhom[x, y]
```

At this point the **simplify** flag can be set again.

```
In[17]:= simplify = True;
```

From the equation for **binhom[x,y]** one readily derives these inclusions:

```
In[18]:= SubstTest[subclass, intersection[u, v], u,
           {u -> fix[composite[inverse[IMAGE[cross[inverse[x], Id]]],
               IMAGE[cross[Id, y]], CROSS, DUP]], v -> map[fix[domain[x]], fix[domain[y]]]}]

Out[18]= subclass[binhom[x, y], fix[composite[
             inverse[IMAGE[cross[inverse[x], Id]]], IMAGE[cross[Id, y]], CROSS, DUP]]] == True

In[19]:= subclass[binhom[x_, y_], fix[composite[
             inverse[IMAGE[cross[inverse[x_], Id]]], IMAGE[cross[Id, y_]], CROSS, DUP]]] := True

In[20]:= SubstTest[subclass, intersection[u, v], v,
           {u -> fix[composite[inverse[IMAGE[cross[inverse[x], Id]]],
               IMAGE[cross[Id, y]], CROSS, DUP]], v -> map[fix[domain[x]], fix[domain[y]]]}]

Out[20]= subclass[binhom[x, y], map[fix[domain[x]], fix[domain[y]]]] == True

In[21]:= subclass[binhom[x_, y_], map[fix[domain[x_]], fix[domain[y_]]]] := True
```

characterizing binary homs

In this section some corollaries of the results of the preceding section are derived that characterize binary homomorphisms. These results will facilitate deriving theorems about binary homomorphisms without having to repeatedly access the **class**-wrapped definition. The following corollary follows from the first inclusion derived in the preceding section.

```
In[24]:= SubstTest[implies, and[member[w, u], subclass[u, v]], member[w, v],
          {u -> binhom[x, y], v -> fix[composite[
              inverse[IMAGE[cross[inverse[x], Id]]], IMAGE[cross[Id, y]], CROSS, DUP]]}]

Out[24]= or[equal[composite[w, x], composite[y, cross[w, w]]],
           not[member[w, binhom[x, y]]]] == True

In[25]:= or[equal[composite[w_, x_], composite[y_, cross[w_, w_]]],
           not[member[w_, binhom[x_, y_]]]] := True
```

No further rewrite rule is needed for the following corollary.

```
In[22]:= implies[member[w, binhom[x, y]], member[w, map[fix[domain[x]], fix[domain[y]]]]]

Out[22]= True
```

In the opposite direction, one has:

```
In[28]:= Map[implies[#, member[w, binhom[x, y]]] &, SubstTest[member, w, intersection[u, v],
            {u -> fix[composite[inverse[IMAGE[cross[inverse[x], Id]]], IMAGE[cross[Id, y]],
                CROSS, DUP]], v -> map[fix[domain[x]], fix[domain[y]]]}]] // Reverse

Out[28]= or[member[w, binhom[x, y]], not[equal[composite[w, x], composite[y, cross[w, w]]]],
           not[member[w, map[fix[domain[x]], fix[domain[y]]]]]] == True

In[29]:= or[member[w_, binhom[x_, y_]],
           not[equal[composite[w_, x_], composite[y_, cross[w_, w_]]]],
           not[member[w_, map[fix[domain[x_]], fix[domain[y_]]]]]] := True
```

composition of binary homs

In this section an application of the rewrite rules characterizing binary homomorphisms is provided. The following argument shows that the composite of binary homomorphisms is a binary homomorphism. (Comment. Because the theorem has not been broken up into a series of lemmas, this takes about a minute.)

```
In[30]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
           implies[p1, p4], implies[p1, p5], implies[and[p2, p3], p6],
           implies[and[p4, p5], p7], implies[and[p6, p7], p8], not[implies[p1, p8]],
           {p1 → and[member[u, binhom[y, z]], member[v, binhom[x, y]]],
            p2 → member[u, map[fix[domain[y]], fix[domain[z]]]],
            p3 -> member[v, map[fix[domain[x]], fix[domain[y]]]],
            p4 → equal[composite[u, y], composite[z, cross[u, u]]],
            p5 -> equal[composite[v, x], composite[y, cross[v, v]]],
            p6 → member[composite[u, v], map[fix[domain[x]], fix[domain[z]]]],
            p7 -> equal[composite[u, v, x],
              composite[z, cross[composite[u, v], composite[u, v]]]],
            p8 -> member[composite[u, v], binhom[x, z]]}]]

Out[30]= or[member[composite[u, v], binhom[x, z]],
           not[member[u, binhom[y, z]]], not[member[v, binhom[x, y]]]] == True

In[31]:= or[member[composite[u_, v_], binhom[x_, z_]],
           not[member[u_, binhom[y_, z_]]], not[member[v_, binhom[x_, y_]]]] := True
```

The variables **u** and **v** can be eliminated:

```
In[32]:= Map[equal[0, composite[Id, complement[#]]] &, SubstTest[class, pair[u, v],
            or[member[composite[u, v], r], not[member[u, s]], not[member[v, t]]],
            {r → binhom[x, z], s -> binhom[y, z], t -> binhom[x, y]}]] // Reverse

Out[32]= subclass[image[COMPOSE, cart[binhom[y, z], binhom[x, y]]], binhom[x, z]] == True

In[33]:= subclass[image[COMPOSE, cart[binhom[y_, z_], binhom[x_, y_]]], binhom[x_, z_]] := True
```

reify rule

A reify rule for the binary functor **binhom** follows immediately from the definition:

```
In[34]:= SubstTest[reify, x, intersection[
           fix[composite[inverse[IMAGE[cross[inverse[f[x]], Id]]], IMAGE[cross[Id, g[x]]], z]],
           map[fix[domain[f[x]]], fix[domain[g[x]]]], z → composite[CROSS, DUP]]

Out[34]= reify[x, binhom[f[x], g[x]]] ==
          composite[id[FUNS], intersection[composite[inverse[IMAGE[FIRST]],
             VERTSECT[composite[inverse[DUP], FIRST, reify[x, f[x]]]], composite[SECOND,
             intersection[composite[cross[Id, composite[inverse[DUP], inverse[CROSS]]],
               inverse[rotate[composite[IMAGE[composite[SWAP, RIF, cross[reify[x, g[x]], Id]]],
                  CART, SWAP]]], inverse[rotate[composite[IMAGE[
                  composite[RIF, cross[composite[SWAP, reify[x, f[x]]], SWAP]]], CART, SWAP]]],
             SINGLETON], composite[inverse[IMAGE[SECOND]], inverse[IMAGE[DUP]],
             inverse[LB[composite[FIRST, reify[x, g[x]]]]]]]]
```

```
In[35]:= reify[x_, binhom[y_, z_]] :=
          composite[id[FUNS], intersection[composite[inverse[IMAGE[FIRST]],
             VERTSECT[composite[inverse[DUP], FIRST, reify[x, y]]], composite[SECOND,
             intersection[composite[cross[Id, composite[inverse[DUP], inverse[CROSS]]],
               inverse[rotate[composite[IMAGE[composite[SWAP, RIF, cross[reify[x, z], Id]]],
                  CART, SWAP]]], inverse[rotate[composite[IMAGE[
                  composite[RIF, cross[composite[SWAP, reify[x, y]], SWAP]]], CART, SWAP]]]],
             SINGLETON], composite[inverse[IMAGE[SECOND]], inverse[IMAGE[DUP]],
             inverse[LB[composite[FIRST, reify[x, z]]]]]]]
```