binary homomorphisms and neutral elements

Johan G. F. Belinfante
2013 August 9

```
In[1]:= SetDirectory["l:"]; << goedel.13aug08a

        :Package Title: goedel.13aug08a                  2013 August 8 at 5:50 p.m.

        Loading takes about seventeen minutes, half that time due to builtin pauses.

        It is now:  2013 Aug 9 at 19:32

        Loading Simplification Rules

        TOOLS.M is now incorporated in the GOEDEL program as of 2010 September 3

        weightlimit = 40

        Loading completed.

        It is now:  2013 Aug 9 at 19:49
```

summary

In general binary homomorphisms need not preserve neutral elements. That is, if **t ∈ binop[x, y]** and if **x** has a neutral element **e[x]**, it need not be true that **APPLY[t, e[x]]** is a neutral element of **y**. In fact **y** need not even have a neutral element. In this notebook it is shown that if a binary operation **x** has a neutral element, then any binary homomorphism **t ∈ binop[x, y]** takes it to the neutral element of its homomorphic image. In other words, **APPLY[t, e[x]]** is a neutral element of the restriction **y ∘ id[range[t] × range[t]]**.

derivation

For convenience the variable for a binary homomorphism will initially be wrapped with **funpart**, but this wrapper is not needed, and will later be eliminated.

Lemma. A restriction of left multiplication by **APPLY[t, e[x]]** is an identity function.

```
In[2]:= SubstTest[implies, and[equal[u, v], subclass[composite[u, w], Id]],
          subclass[composite[v, w], Id], {u -> composite[funpart[t], binop[x]],
           v -> composite[binop[y], cross[funpart[t], funpart[t]]],
           w → composite[LEFT[e[binop[x]]], inverse[funpart[t]]]}] // Reverse

Out[2]= or[not[equal[composite[binop[y], cross[funpart[t], funpart[t]]],
            composite[funpart[t], binop[x]]]], subclass[composite[binop[y],
            LEFT[APPLY[funpart[t], e[binop[x]]]], id[range[funpart[t]]]], Id]] == True
```

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

Lemma. A restriction of right multiplication by **APPLY[t, e[x]]** is an identity function.

```
In[4]:= SubstTest[implies, and[equal[u, v], subclass[composite[u, w], Id]],
          subclass[composite[v, w], Id], {u -> composite[funpart[t], binop[x]],
           v -> composite[binop[y], cross[funpart[t], funpart[t]]],
           w → composite[RIGHT[e[binop[x]]], inverse[funpart[t]]]}] // Reverse

Out[4]= or[not[equal[composite[binop[y], cross[funpart[t], funpart[t]]],
            composite[funpart[t], binop[x]]]], subclass[composite[binop[y],
            RIGHT[APPLY[funpart[t], e[binop[x]]]], id[range[funpart[t]]]], Id]] == True
```

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

Lemma.

```
In[6]:= Map[not, SubstTest[and, implies[p1, p4], implies[p2, p3],
           implies[and[p3, p4], p5], not[implies[and[p1, p2], p5]],
           {p1 → member[e[binop[x]], V], p2 → member[t, binhom[binop[x], binop[y]]],
            p3 → equal[domain[t], fix[domain[binop[x]]]],
            p4 → member[e[binop[x]], fix[domain[binop[x]]]],
            p5 → member[e[binop[x]], domain[t]]}]] // Reverse

Out[6]= or[member[e[binop[x]], domain[t]],
          not[member[t, binhom[binop[x], binop[y]]]], not[member[e[binop[x]], V]]] == True
```

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

Lemma. A condition for **u** to be a neutral element for a restriction of **x** to a cartesian square **t** × **t**.

```
In[8]:= Map[implies[#, member[u, ids[composite[x, id[cart[t, t]]]]]] &, SubstTest[and,
           subclass[composite[w, LEFT[u]], Id], subclass[composite[w, RIGHT[u]], Id],
           member[u, fix[domain[w]]], w → composite[x, id[cart[t, t]]]]] // Reverse

Out[8]= or[member[u, ids[composite[x, id[cart[t, t]]]]], not[member[u, t]],
          not[member[u, fix[domain[x]]]], not[subclass[composite[x, LEFT[u], id[t]], Id]],
          not[subclass[composite[x, RIGHT[u], id[t]], Id]]] == True
```

```
In[9]:= (% /. {t → t_, u → u_, x → x_}) /. Equal → SetDelayed
```

Lemma. A sufficient condition for **APPLY[t, e[x]]** to be a neutral element of the restriction of **y** to **range[t]** × **range[t]**.

```
In[10]:= SubstTest[or, member[u, ids[composite[w, id[cart[s, s]]]]], not[member[u, s]],
           not[member[u, fix[domain[w]]]], not[subclass[composite[w, LEFT[u], id[s]], Id]],
           not[subclass[composite[w, RIGHT[u], id[s]], Id]],
           {w → binop[y], s → range[funpart[t]], u -> APPLY[funpart[t], e[binop[x]]]}] // Reverse

Out[10]= or[member[APPLY[funpart[t], e[binop[x]]],
            ids[composite[binop[y], id[cart[range[funpart[t]], range[funpart[t]]]]]]],
           not[member[APPLY[funpart[t], e[binop[x]]], fix[domain[binop[y]]]]],
           not[member[e[binop[x]], domain[funpart[t]]]],
           not[subclass[composite[binop[y], LEFT[APPLY[funpart[t], e[binop[x]]]],
              id[range[funpart[t]]], Id]], not[subclass[composite[binop[y],
              RIGHT[APPLY[funpart[t], e[binop[x]]]], id[range[funpart[t]]], Id]]] == True

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

Each of the hypotheses of the above lemma will be examined in turn. The starting point is a general fact.

Theorem. A general fact.

```
In[12]:= Map[not, SubstTest[and, implies[p1, p3], implies[and[p2, p3], p4],
            not[implies[and[p1, p2], p4]], {p1 → member[t, binhom[x, y]],
             p2 → member[u, fix[domain[x]]], p3 → member[t, map[fix[domain[x]], fix[domain[y]]]],
             p4 -> member[APPLY[t, u], fix[domain[y]]]}]] // Reverse

Out[12]= or[member[APPLY[t, u], fix[domain[y]]],
           not[member[t, binhom[x, y]]], not[member[u, fix[domain[x]]]]] == True

In[13]:= or[member[APPLY[t_, u_], fix[domain[y_]]],
           not[member[u_, fix[domain[x_]]]], not[member[t_, binhom[x_, y_]]]] := True
```

Specializing to the case **u = e[x]** yields this corollary.

Corollary. If a binary operation **x** has a neutral element **e[x]**, then any binary homomorphism **t ∈ binhom[x, y]** takes **e[x]** to an element of **fix[domain[y]]**.

```
In[14]:= Map[implies[member[e[binop[x]], z], #] &, SubstTest[implies,
            and[member[t, binhom[binop[x], y]], member[u, fix[domain[binop[x]]]]],
            member[APPLY[t, u], fix[domain[y]]], u → e[binop[x]]]] // Reverse

Out[14]= or[member[APPLY[t, e[binop[x]]], fix[domain[y]]],
           not[member[t, binhom[binop[x], y]]], not[member[e[binop[x]], z]]] == True

In[15]:= or[member[APPLY[t_, e[binop[x_]]], fix[domain[y_]]],
           not[member[e[binop[x_]], z_]], not[member[t_, binhom[binop[x_], y_]]]] := True
```

Theorem.

```
In[16]:= Map[not, SubstTest[and, implies[p1, p2], (*implies[p2,p3],
             implies[p2,p4],implies[and[p0,p1],p5],implies[and[p0,p1],p6],*)
            implies[and[p0, p3, p4, p5, p6], p7], not[implies[and[p0, p1], p7]],
            {p0 → member[e[binop[x]], V], p1 → member[funpart[t], binhom[binop[x], binop[y]]],
             p2 → equal[composite[funpart[t], binop[x]],
               composite[binop[y], cross[funpart[t], funpart[t]]]],
             p3 -> subclass[composite[binop[y], LEFT[APPLY[funpart[t], e[binop[x]]]],
                id[range[funpart[t]]]], Id], p4 -> subclass[composite[binop[y],
                RIGHT[APPLY[funpart[t], e[binop[x]]]], id[range[funpart[t]]]], Id],
             p5 → member[APPLY[funpart[t], e[binop[x]]], range[funpart[t]]],
             p6 -> member[APPLY[funpart[t], e[binop[x]]], fix[domain[binop[y]]]],
             p7 -> member[APPLY[funpart[t], e[binop[x]]],
               ids[composite[binop[y], id[cartsq[range[funpart[t]]]]]]]}]] // Reverse

Out[16]= or[member[APPLY[funpart[t], e[binop[x]]],
            ids[composite[binop[y], id[cart[range[funpart[t]], range[funpart[t]]]]]], not[
            member[e[binop[x]], V]], not[member[funpart[t], binhom[binop[x], binop[y]]]]] == True

In[17]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. Binary operations have at most one neutral element.

```
In[18]:= SubstTest[implies, and[member[u, ids[v]], member[v, BINOPS]],
           equal[u, e[v]], {u -> APPLY[funpart[t], e[binop[x]]],
            v -> composite[binop[y], id[cart[range[funpart[t]], range[funpart[t]]]]]}] // Reverse

Out[18]= or[equal[APPLY[funpart[t], e[binop[x]]],
            e[composite[binop[y], id[cart[range[funpart[t]], range[funpart[t]]]]]],
           not[member[APPLY[funpart[t], e[binop[x]]],
             ids[composite[binop[y], id[cart[range[funpart[t]], range[funpart[t]]]]]]],
           not[subclass[image[binop[y], cart[range[funpart[t]], range[funpart[t]]]],
             range[funpart[t]]]]] == True

In[19]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[20]:= Map[not, SubstTest[and, (*implies[p1,p2],implies[and[p0,p1],p3],
             implies[p1,p4], *)implies[and[p2, p3, p4], p5], not[implies[and[p0, p1], p5]],
            {p0 → member[e[binop[x]], V], p1 → member[funpart[t], binhom[binop[x], binop[y]]],
             p2 → member[composite[binop[y], id[cart[range[funpart[t]], range[funpart[t]]]]],
               BINOPS], p3 -> member[APPLY[funpart[t], e[binop[x]]],
               ids[composite[binop[y], id[cart[range[funpart[t]], range[funpart[t]]]]]],
             p4 -> subclass[image[binop[y], cart[range[funpart[t]], range[funpart[t]]]],
               range[funpart[t]]], p5 → equal[APPLY[funpart[t], e[binop[x]]], e[composite[
                 binop[y], id[cart[range[funpart[t]], range[funpart[t]]]]]]]}]] // Reverse

Out[20]= or[equal[APPLY[funpart[t], e[binop[x]]],
            e[composite[binop[y], id[cart[range[funpart[t]], range[funpart[t]]]]]], not[
            member[e[binop[x]], V]], not[member[funpart[t], binhom[binop[x], binop[y]]]]] == True

In[21]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

The **funpart** wrapper can be eliminated. In the next section it will be shown that the binop wrapper on **y** is unnecessary as well.

Theorem. If a binary operation has a neutral element, then any binary homomorphism takes it to the neutral element of its homomorphic image.

```
In[22]:= Map[implies[member[e[binop[x]], z], #] &,
           SubstTest[implies, equal[t, funpart[s]], or[equal[APPLY[t, e[binop[x]]],
              e[composite[binop[y], id[cart[range[t], range[t]]]]]],
             not[member[e[binop[x]], V]], not[member[t, binhom[binop[x], binop[y]]]]],
            s → t]] // MapNotNot // Reverse

Out[22]= or[equal[APPLY[t, e[binop[x]]], e[composite[binop[y], id[cart[range[t], range[t]]]]]],
           not[member[t, binhom[binop[x], binop[y]]]], not[member[e[binop[x]], z]]] == True

In[23]:= (% /. {t → t_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Corollary. If a binary operation has a neutral element, then it is preserved by an onto binary homomorphism.

```
In[24]:= Map[implies[equal[fix[domain[binop[y]]], range[t]], #] &,
           SubstTest[implies, equal[w, composite[binop[y], id[cart[range[t], range[t]]]]],
            or[equal[APPLY[t, e[binop[x]]], e[w]], not[member[t, binhom[binop[x], binop[y]]]],
             not[member[e[binop[x]], z]]], w → binop[y]]] // Reverse

Out[24]= or[equal[APPLY[t, e[binop[x]]], e[binop[y]]],
           not[equal[fix[domain[binop[y]]], range[t]]],
           not[member[t, binhom[binop[x], binop[y]]]], not[member[e[binop[x]], z]]] == True

In[25]:= or[equal[APPLY[t_, e[binop[x_]]], e[binop[y_]]],
           not[equal[fix[domain[binop[y_]]], range[t_]]], not[member[e[binop[x_]], z_]],
           not[member[t_, binhom[binop[x_], binop[y_]]]]] := True
```

a slight improvement

In the final theorem of the last section, the hypothesis that **y** be a binary operation is not needed. If **t ∈ binhom[x, y]** and if **x** is a binary operation, then its homomorphic image is a binary homomorphism.

Lemma. (Remove the **binop** wrappers.)

```
In[26]:= SubstTest[implies, and[equal[x, binop[u]], equal[y, binop[v]]],
           or[equal[APPLY[t, e[x]], e[y]], not[equal[fix[domain[y]], range[t]]],
            not[member[t, binhom[x, y]]], not[member[e[x], V]]], {u → x, v → y}] // Reverse

Out[26]= or[equal[APPLY[t, e[x]], e[y]],
           not[equal[fix[domain[y]], range[t]]], not[member[t, binhom[x, y]]],
           not[member[x, BINOPS]], not[member[y, BINOPS]], not[member[e[x], V]]] == True

In[27]:= or[equal[APPLY[t_, e[x_]], e[y_]], not[equal[fix[domain[y_]], range[t_]]],
           not[member[e[x_], V]], not[member[t_, binhom[x_, y_]]],
           not[member[x_, BINOPS]], not[member[y_, BINOPS]]] := True
```

Lemma. Specialize to the case **y** is the homomorphic image of **x** under **t**.

```
In[28]:= SubstTest[or, equal[APPLY[t, e[x]], e[z]], not[equal[fix[domain[z]], range[t]]],
           not[member[t, binhom[x, z]]], not[member[x, BINOPS]], not[member[z, BINOPS]],
           not[member[e[x], V]], z → composite[y, id[cart[range[t], range[t]]]]] // Reverse

Out[28]= or[equal[APPLY[t, e[x]], e[composite[y, id[cart[range[t], range[t]]]]]],
           not[member[t, binhom[x, y]]], not[member[x, BINOPS]],
           not[member[composite[y, id[cart[range[t], range[t]]]], BINOPS]],
           not[member[e[x], V]], not[subclass[range[t], fix[domain[y]]]] == True

In[29]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem.

```
In[30]:= Map[implies[member[e[x], z], not[#]] &, SubstTest[and,
            implies[and[p1, p3], p4], implies[p3, p5], not[implies[and[p1, p2, p3], p6]],
            {p1 → member[x, BINOPS], p2 → member[e[x], V], p3 -> member[t, binhom[x, y]],
             p4 -> member[composite[y, id[cart[range[t], range[t]]]], BINOPS],
             p5 -> subclass[range[t], fix[domain[y]]], p6 ->
              equal[APPLY[t, e[x]], e[composite[y, id[cart[range[t], range[t]]]]]]}]] // Reverse

Out[30]= or[equal[APPLY[t, e[x]], e[composite[y, id[cart[range[t], range[t]]]]]],
           not[member[t, binhom[x, y]]], not[member[x, BINOPS]], not[member[e[x], z]]] == True

In[31]:= or[equal[APPLY[t_, e[x_]], e[composite[y_, id[cart[range[t_], range[t_]]]]]],
           not[member[e[x_], z_]], not[member[t_, binhom[x_, y_]]],
           not[member[x_, BINOPS]]] := True
```

The following is a slight improvement over the theorem derived in the preceding section.

Corollary. (Reintroduce the **binop** wrapper.)

```
In[32]:= SubstTest[or, equal[APPLY[t, e[w]], e[composite[y, id[cart[range[t], range[t]]]]]],
           not[member[t, binhom[w, y]]], not[member[w, BINOPS]],
           not[member[e[w], z]], w → binop[x]] // Reverse

Out[32]= or[equal[APPLY[t, e[binop[x]]], e[composite[y, id[cart[range[t], range[t]]]]]],
           not[member[t, binhom[binop[x], y]]], not[member[e[binop[x]], z]]] == True

In[33]:= or[equal[APPLY[t_, e[binop[x_]]], e[composite[y_, id[cart[range[t_], range[t_]]]]]],
           not[member[e[binop[x_]], z_]], not[member[t_, binhom[binop[x_], y_]]]] := True
```

One also needs to know that **e[y ∘ id[range[t] × range[t]]]** is a set.

Theorem. If a binary operation has a neutral element, then so do its homomorphic images.

```
In[34]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[and[p2, p3], p4],
           implies[and[p1, p2], p5], implies[and[p4, p5], p6], not[implies[and[p1, p2], p6]],
           {p1 → member[t, binhom[binop[x], y]], p2 -> member[e[binop[x]], z],
            p3 → equal[domain[t], fix[domain[binop[x]]]], p4 → member[APPLY[t, e[binop[x]]], V],
            p5 -> equal[APPLY[t, e[binop[x]]], e[composite[y, id[cart[range[t], range[t]]]]]],
            p6 → member[e[composite[y, id[cart[range[t], range[t]]]]], V]}]] // Reverse

Out[34]= or[member[e[composite[y, id[cart[range[t], range[t]]]]], V],
           not[member[t, binhom[binop[x], y]]], not[member[e[binop[x]], z]]] == True

In[35]:= or[member[e[composite[y_, id[cart[range[t_], range[t_]]]]], V],
           not[member[t_, binhom[binop[x_], y_]]], not[member[e[binop[x_]], z_]]] := True
```

Corollary. (Replace the **binop** wrapper with a membership literal.)

```
In[36]:= SubstTest[implies, equal[x, binop[w]],
           or[member[e[composite[y, id[cart[range[t], range[t]]]]], V],
            not[member[t, binhom[x, y]]], not[member[e[x], z]]], w → x] // Reverse

Out[36]= or[member[e[composite[y, id[cart[range[t], range[t]]]]], V],
           not[member[t, binhom[x, y]]], not[member[x, BINOPS]], not[member[e[x], z]]] == True

In[37]:= or[member[e[composite[y_, id[cart[range[t_], range[t_]]]]], V],
           not[member[t_, binhom[x_, y_]]],
           not[member[x_, BINOPS]], not[member[e[x_], z_]]] := True
```

---

homomorphic images of monoids

An important special case is treated in this section. To reduce execution time, some hypotheses are commented out. The following derivation takes 12 seconds.

Theorem. The homomorphic image of a monoid is a monoid.

```
In[38]:= Map[not,
           SubstTest[and, (*implies[p1,p3],implies[p1,p4],*) implies[and[p2, p3, p4], p5],
            (*implies[p1,p6],*) implies[and[p2, p6], p7], (* implies[and[p5,p7],p8],*)
            not[implies[and[p1, p2], p8]], {p1 → member[x, MONOIDS],
             p2 → member[t, binhom[x, y]], p3 → member[e[x], V], p4 → member[x, BINOPS], p5 ->
              member[e[composite[y, id[cart[range[t], range[t]]]]], V], p6 → member[x, SEMIGPS],
             p7 -> member[composite[y, id[cart[range[t], range[t]]]], SEMIGPS],
             p8 -> member[composite[y, id[cart[range[t], range[t]]]], MONOIDS]}]] // Reverse

Out[38]= or[member[composite[y, id[cart[range[t], range[t]]]], MONOIDS],
           not[member[t, binhom[x, y]]], not[member[x, MONOIDS]]] == True

In[39]:= or[member[composite[y_, id[cart[range[t_], range[t_]]]], MONOIDS],
           not[member[t_, binhom[x_, y_]]], not[member[x_, MONOIDS]]] := True
```