group binhoms preserve neutral elements

Johan G. F. Belinfante
2011 December 17; revised 2011 December 31

```
In[1]:= SetDirectory["l:"]; << goedel.11dec16a

        :Package Title: goedel.11dec16a              2011 December 16 at 3:00 p.m.

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

        It is now:  2011 Dec 31 at 23:7

        Loading Simplification Rules

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

        weightlimit = 40

        Loading completed.

        It is now:  2011 Dec 31 at 23:20
```

summary

Since binary homomorphisms preserve idempotents, and since the neutral element of a group is its only idempotent, it follows that binary homomorphisms between groups preserve neutral elements.

mapping statements for binhoms

The term **group** in the **GOEDEL** program refers to a binary composition law; the underlying set on which it acts is its range. For a group, the domain is the cartesian square of the range, and hence **fix[domain[x]] = range[x]**. This simplification is done automatically when one uses the **gp** wrapper. (In general **gp[x]** is either a group or the empty set.)

Theorem. A binary homomorphism from **gp[x]** to **gp[y]** is a mapping from the range of one to the range of the other.

```
In[2]:= SubstTest[implies, member[t, binhom[u, v]],
          member[t, map[fix[domain[u]], fix[domain[v]]]], {u → gp[x], v → gp[y]}] // Reverse

Out[2]= or[member[t, map[range[gp[x]], range[gp[y]]]],
          not[member[t, binhom[gp[x], gp[y]]]]] == True

In[3]:= or[member[t_, map[range[gp[x_]], range[gp[y_]]]],
          not[member[t_, binhom[gp[x_], gp[y_]]]]] := True
```

In removing the **gp** wrappers, the use of **MapNotNot** gets rid of the nuisance cases involving empty composition laws.

Corollary. A binary homomorphism from one group to another is a mapping from the range of the one to the range of the other.

```
In[4]:= SubstTest[implies, and[equal[x, gp[u]], equal[y, gp[v]]],
          or[member[t, map[range[x], range[y]]], not[member[t, binhom[x, y]]]],
          {u → x, v → y}] // Reverse // MapNotNot

Out[4]= or[member[t, map[range[x], range[y]]], not[member[t, binhom[x, y]]],
          not[member[x, GROUPS]], not[member[y, GROUPS]]] == True

In[5]:= or[member[t_, map[range[x_], range[y_]]], not[member[t_, binhom[x_, y_]]],
          not[member[x_, GROUPS]], not[member[y_, GROUPS]]] := True
```

In automated reasoning, a common strategy is to focus on clauses consisting of a single literal. The result derived in this section can be stated using a single literal by retaining the **gp** wrappers, and instead eliminating the variable **t**.

Corollary. The set of binary homomorphisms from **gp[x]** to **gp[y]** is a subset of the set of mappings from **range[gp[x]]** to **range[gp[y]]**.

```
In[6]:= subclass[binhom[gp[x], gp[y]], map[range[gp[x]], range[gp[y]]]] // AssertTest

Out[6]= subclass[binhom[gp[x], gp[y]], map[range[gp[x]], range[gp[y]]]] == True

In[7]:= subclass[binhom[gp[x_], gp[y_]], map[range[gp[x_]], range[gp[y_]]]] := True
```

The definition of binary homorphism can be specialized to group theory by introducing **gp** wrappers.

Theorem. A sufficient condition for a mapping from **gp[x]** to **gp[y]** to be a binary homomorphism.

```
In[8]:= SubstTest[implies, and[equal[composite[t, u], composite[v, cross[t, t]]],
           member[t, map[fix[domain[u]], fix[domain[v]]]]],
          member[t, binhom[u, v]], {u → gp[x], v → gp[y]}] // Reverse

Out[8]= or[member[t, binhom[gp[x], gp[y]]],
          not[equal[composite[t, gp[x]], composite[gp[y], cross[t, t]]]],
          not[member[t, map[range[gp[x]], range[gp[y]]]]]] == True

In[9]:= or[member[t_, binhom[gp[x_], gp[y_]]],
          not[equal[composite[gp[y_], cross[t_, t_]], composite[t_, gp[x_]]]],
          not[member[t_, map[range[gp[x_]], range[gp[y_]]]]]] := True
```

binhoms preserve neutral elements

Binary homomorphisms preserve idempotents, and the neutral element of a group is its only idempotent. When using the **gp[x]** wrapper, of course, there might not be any neutral element, in which case the singleton of **e[gp[x]]** is empty.

Lemma. A binary homomorphism from **gp[x]** to **gp[y]** preserves neutral elements if any.

```
In[10]:= SubstTest[implies, member[w, binhom[u, v]],
           subclass[image[w, fix[composite[u, DUP]]], fix[composite[v, DUP]]],
           {u → gp[x], v → gp[y]}] // Reverse

Out[10]= or[not[member[w, binhom[gp[x], gp[y]]]],
           subclass[image[w, set[e[gp[x]]]], set[e[gp[y]]]]] == True

In[11]:= or[not[member[w_, binhom[gp[x_], gp[y_]]]],
           subclass[image[w_, set[e[gp[x_]]]], set[e[gp[y_]]]]] := True
```

Lemma. (Introducing a **funpart** wrapper causes the image to be rewritten in terms of function application.)

```
In[12]:= SubstTest[implies, member[w, binhom[gp[x], gp[y]]],
           subclass[image[w, set[e[gp[x]]]], set[e[gp[y]]]], w → funpart[t]] // Reverse

Out[12]= or[equal[APPLY[funpart[t], e[gp[x]]], e[gp[y]]],
           not[member[e[gp[x]], domain[funpart[t]]]],
           not[member[funpart[t], binhom[gp[x], gp[y]]]]] == True

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

Lemma. (Remove all wrappers.)

```
In[14]:= SubstTest[implies, and[equal[t, funpart[w]], equal[x, gp[u]], equal[y, gp[v]]],
            or[equal[APPLY[t, e[x]], e[y]], not[member[e[x], domain[t]]],
             not[member[t, binhom[x, y]]]], {w → t, u → x, v → y}] // Reverse // MapNotNot

Out[14]= or[equal[APPLY[t, e[x]], e[y]], not[FUNCTION[t]], not[member[t, binhom[x, y]]],
           not[member[x, GROUPS]], not[member[y, GROUPS]], not[member[e[x], domain[t]]]] == True

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

The final step is to clean this up by removing two redundant literals. To expedite the derivation, two proof steps are omitted, as indicated by **(* ⋯ *)**.

Theorem. Binary homomorphisms from one group to another preserve neutral elements.

```
In[16]:= Map[not, SubstTest[and, (*implies[and[p1,p2,p3],p4],*) implies[p3, p5],
            implies[p4, p6], implies[and[p1, p4, p6], p7], (*implies[and[p1,p2,p3,p5,p7],p8],*)
            not[implies[and[p1, p2, p3], p8]], {p1 → member[x, GROUPS], p2 → member[y, GROUPS],
             p3 -> member[t, binhom[x, y]], p4 → member[t, map[range[x], range[y]]],
             p5 → FUNCTION[t], p6 → equal[domain[t], range[x]],
             p7 -> member[e[x], domain[t]], p8 -> equal[APPLY[t, e[x]], e[y]]}]] // Reverse

Out[16]= or[equal[APPLY[t, e[x]], e[y]], not[member[t, binhom[x, y]]],
           not[member[x, GROUPS]], not[member[y, GROUPS]]] == True

In[17]:= or[equal[APPLY[t_, e[x_]], e[y_]], not[member[t_, binhom[x_, y_]]],
           not[member[x_, GROUPS]], not[member[y_, GROUPS]]] := True
```