binary homs preserve commutativity

Johan G. F. Belinfante
2013 July 5

```
In[1]:= SetDirectory["l:"]; << goedel.13jul04a

        :Package Title: goedel.13jul04a                    2013 July 4 at 5:25 p.m.

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

        It is now:  2013 Jul 5 at 17:44

        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 Jul 5 at 18:0
```

summary

Binary homomorphisms of binary operations preserve their commutativity relations. A recent result about the regular representation of a semigroup is rederived in another way as an application.

derivation

Lemma. A basic monotonicity result.

```
In[2]:= SubstTest[implies, and[subclass[u, v], subclass[x, y]],
          subclass[composite[u, x], composite[v, y]], {u → inverse[x], v → inverse[y]}] // Reverse

Out[2]= or[not[subclass[x, y]],
          subclass[composite[inverse[x], x], composite[inverse[y], y]]] == True

In[4]:= or[not[subclass[x_, y_]],
          subclass[composite[inverse[x_], x_], composite[inverse[y_], y_]]] := True
```

Lemma. Application to binary homomorphisms.

```
In[5]:= SubstTest[implies, equal[u, v],
          equal[composite[inverse[u], u], composite[inverse[v], v]],
          {u -> composite[t, x], v -> composite[y, cross[t, t]]}] // Reverse

Out[5]= or[equal[composite[cross[inverse[t], inverse[t]], inverse[y], y, cross[t, t]],
           composite[inverse[x], inverse[t], t, x]],
          not[equal[composite[t, x], composite[y, cross[t, t]]]]] == True

In[6]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. An inclusion.

```
In[7]:= SubstTest[implies, subclass[u, v],
          subclass[composite[inverse[x], u, x], composite[inverse[x], v, x]],
          {u -> id[range[x]], v -> composite[inverse[t], t]}] // Reverse

Out[7]= or[not[subclass[range[x], domain[t]]],
          subclass[composite[inverse[x], x], composite[inverse[x], inverse[t], t, x]]] == True

In[8]:= (% /. {t → t_, x → x_}) /. Equal → SetDelayed
```

Lemma. Temporary simplification rule involving the **funpart** wrapper.

```
In[9]:= SubstTest[fix, composite[inverse[funpart[w]], z, funpart[w]],
          {w → cross[funpart[t], funpart[t]], z → composite[SWAP, x]}] // Reverse

Out[9]= fix[composite[SWAP,
           cross[inverse[funpart[t]], inverse[funpart[t]]], x, cross[funpart[t], funpart[t]]]] ==
         composite[inverse[funpart[t]], fix[composite[SWAP, x]], funpart[t]]

In[10]:= fix[composite[SWAP, cross[inverse[funpart[t_]], inverse[funpart[t_]]],
            x_, cross[funpart[t_], funpart[t_]]]] :=
          composite[inverse[funpart[t]], fix[composite[SWAP, x]], funpart[t]]
```

Lemma. Application to the commutativity relation.

```
In[11]:= SubstTest[implies, subclass[u, v],
           subclass[fix[coflip[u]], fix[coflip[v]]], {u -> composite[inverse[x], x],
            v -> composite[cross[inverse[funpart[t]], inverse[funpart[t]]],
              inverse[y], y, cross[funpart[t], funpart[t]]]}] // Reverse

Out[11]= or[not[subclass[composite[inverse[x], x],
             composite[cross[inverse[funpart[t]], inverse[funpart[t]]],
              inverse[y], y, cross[funpart[t], funpart[t]]]]],
            subclass[fix[composite[SWAP, inverse[x], x]], composite[inverse[funpart[t]],
              fix[composite[SWAP, inverse[y], y]], funpart[t]]]] == True

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

Lemma. Putting all the pieces together.

```
In[13]:= Map[not, SubstTest[and, (*implies[p1,p2],implies[p1,p3],*) implies[p3, p4],
           implies[p2, p5], (*implies[and[p4,p5],p6],*) implies[p6, p7], not[implies[p1, p7]],
           {p1 → and[member[x, BINOPS], member[funpart[t], binhom[x, y]]],
            p2 → equal[composite[y, cross[funpart[t], funpart[t]]], composite[funpart[t], x]],
            p3 → subclass[range[x], domain[funpart[t]]], p4 → subclass[composite[inverse[x], x],
              composite[inverse[x], inverse[funpart[t]], funpart[t], x]],
            p5 → equal[composite[cross[inverse[funpart[t]], inverse[funpart[t]]],
                inverse[y], y, cross[funpart[t], funpart[t]]],
              composite[inverse[x], inverse[funpart[t]], funpart[t], x]],
            p6 → subclass[composite[inverse[x], x], composite[cross[inverse[funpart[t]],
                inverse[funpart[t]]], inverse[y], y, cross[funpart[t], funpart[t]]]],
            p7 → subclass[fix[composite[SWAP, inverse[x], x]], composite[inverse[funpart[t]],
                fix[composite[SWAP, inverse[y], y]], funpart[t]]]}]] // Reverse

Out[13]= or[not[member[x, BINOPS]], not[member[funpart[t], binhom[x, y]]],
           subclass[fix[composite[SWAP, inverse[x], x]], composite[inverse[funpart[t]],
            fix[composite[SWAP, inverse[y], y]], funpart[t]]]] == True

In[14]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. (Eliminating the **funpart** wrapper.)

```
In[15]:= SubstTest[implies, equal[t, funpart[w]],
           or[not[member[x, BINOPS]], not[member[t, binhom[x, y]]],
            subclass[fix[composite[SWAP, inverse[x], x]], composite[inverse[t],
              fix[composite[SWAP, inverse[y], y]], t]]], w → t] // Reverse // MapNotNot

Out[15]= or[not[member[t, binhom[x, y]]],
           not[member[x, BINOPS]], subclass[fix[composite[SWAP, inverse[x], x]],
            composite[inverse[t], fix[composite[SWAP, inverse[y], y]], t]]] == True

In[16]:= or[not[member[t_, binhom[x_, y_]]],
           not[member[x_, BINOPS]], subclass[fix[composite[SWAP, inverse[x_], x_]],
            composite[inverse[t_], fix[composite[SWAP, inverse[y_], y_]], t_]]] := True
```

The variable **t** can be moved to the other side of the inclusion, if desired.

Lemma.

```
In[17]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 -> and[member[x, BINOPS], member[funpart[t], binhom[x, y]]],
            p2 -> subclass[fix[composite[SWAP, inverse[x], x]],
              composite[inverse[funpart[t]], fix[composite[SWAP, inverse[y], y]], funpart[t]]],
            p3 -> subclass[composite[funpart[t], fix[composite[SWAP, inverse[x], x]],
                inverse[funpart[t]]], fix[composite[SWAP, inverse[y], y]]]}]] // Reverse

Out[17]= or[not[member[x, BINOPS]], not[member[funpart[t], binhom[x, y]]],
           subclass[composite[funpart[t], fix[composite[SWAP, inverse[x], x]],
            inverse[funpart[t]]], fix[composite[SWAP, inverse[y], y]]]] == True

In[18]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

Main Theorem. Binary homomorphisms of binary operations preserve the commutativity relation.

```
In[19]:= SubstTest[implies, equal[t, funpart[w]],
           or[not[member[x, BINOPS]], not[member[t, binhom[x, y]]],
            subclass[composite[t, fix[composite[SWAP, inverse[x], x]], inverse[t]],
             fix[composite[SWAP, inverse[y], y]]]], w → t] // Reverse // MapNotNot

Out[19]= or[not[member[t, binhom[x, y]]], not[member[x, BINOPS]],
           subclass[composite[t, fix[composite[SWAP, inverse[x], x]], inverse[t]],
            fix[composite[SWAP, inverse[y], y]]]] == True

In[20]:= or[not[member[t_, binhom[x_, y_]]], not[member[x_, BINOPS]],
           subclass[composite[t_, fix[composite[SWAP, inverse[x_], x_]], inverse[t_]],
            fix[composite[SWAP, inverse[y_], y_]]]] := True
```

As a corollary, a recent result about the regular representation of a semigroup can now be rederived in another way. Because the present derivation did not introduce variables for elements of binary operations, some of the complications encountered in the notebook **cmt-rrep.nb** were avoided.

Corollary. The commutativity relation for a semigroup is related to the relation **COMMUTE** via the regular representation **APPLY[CURRY, semigp[x]]**.

```
In[21]:= SubstTest[implies, and[member[t, binhom[u, v]], member[u, BINOPS]],
           subclass[composite[t, fix[composite[SWAP, inverse[u], u]], inverse[t]],
            fix[composite[SWAP, inverse[v], v]]],
           {t → APPLY[CURRY, semigp[x]], u → semigp[x], v → COMPOSE}] // Reverse

Out[21]= subclass[composite[APPLY[CURRY, semigp[x]],
            fix[composite[SWAP, inverse[semigp[x]], semigp[x]]],
            inverse[APPLY[CURRY, semigp[x]]]], COMMUTE] == True

In[22]:= subclass[composite[APPLY[CURRY, semigp[x_]],
            fix[composite[SWAP, inverse[semigp[x_]], semigp[x_]]],
            inverse[APPLY[CURRY, semigp[x_]]]], COMMUTE] := True
```