binary homomorphisms from INTADD to a group

Johan G. F. Belinfante
2013 February 21

```
In[1]:= SetDirectory["l:"]; << goedel.13feb04a

        :Package Title: goedel.13feb04a              2013 February 4 at 6:00 p.m.

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

        It is now:  2013 Feb 21 at 10:59

        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 Feb 21 at 11:16
```

summary

Any binary homomorphism **t** from **INTADD** to a group **gp[x]** is determined by the element **APPLY[t, id[ω] ∘ SUCC] ∈ range[gp[x]]**.
In this notebook a concise formulation of this is derived that does not involve the variable **t**.

Comment. The set **gp[x]** in general could be either empty or a group. But here, the assumption **t ∈ binhom[INTADD, gp[x]]** implies that **gp[x]** is not empty, so it is indeed a group.

derivation

Theorem.

```
In[2]:= SubstTest[implies, and[member[t, binhom[INTADD, x]], member[u, binhom[NATADD, INTADD]]],
          member[composite[t, u], binhom[NATADD, x]], u → composite[INVERSE, PLUS]] // Reverse

Out[2]= or[member[composite[t, INVERSE, PLUS], binhom[NATADD, x]],
          not[member[t, binhom[INTADD, x]]]] == True

In[3]:= or[member[composite[t_, INVERSE, PLUS], binhom[NATADD, x_]],
          not[member[t_, binhom[INTADD, x_]]]] := True
```

The functions **PLUS** and **INVERSE ∘ PLUS** are binary homomorphisms from natural into integer addition. One can therefore manufacture two binary homomorphisms from **NATADD** to **gp[x]** from any **t ∈ binhom[INTADD, gp[x]]**.

Lemma.

```
In[4]:= SubstTest[implies, member[t, GROUPS],
          or[equal[w, iterate[composite[t, LEFT[APPLY[w, set[0]]]], set[e[t]]]],
           not[member[w, binhom[NATADD, t]]]], t → gp[x]] // Reverse

Out[4]= or[equal[0, gp[x]],
          equal[w, iterate[composite[gp[x], LEFT[APPLY[w, set[0]]]], set[e[gp[x]]]]],
          not[member[w, binhom[NATADD, gp[x]]]] == True

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

The first literal is redundant.

Theorem. Any binary homomorphism from **NATADD** to **gp[x]** is a list of powers.

```
In[6]:= SubstTest[and, implies[p, q], or[p, q], {p -> equal[0, gp[x]],
          q -> or[equal[w, iterate[composite[gp[x], LEFT[APPLY[w, set[0]]]], set[e[gp[x]]]]],
            not[member[w, binhom[NATADD, gp[x]]]]}]

Out[6]= or[equal[w, iterate[composite[gp[x], LEFT[APPLY[w, set[0]]]], set[e[gp[x]]]]],
          not[member[w, binhom[NATADD, gp[x]]]] == True

In[7]:= or[equal[iterate[composite[gp[x_], LEFT[APPLY[w_, set[0]]]], set[e[gp[x_]]]], w_],
          not[member[w_, binhom[NATADD, gp[x_]]]]] := True
```

The above result can now be specialized to the two binary homomorphisms **t ∘ PLUS** and **t ∘ INVERSE ∘ PLUS**. This first case is easy.

Lemma.

```
In[8]:= SubstTest[or,
          equal[w, iterate[composite[gp[x], LEFT[APPLY[w, set[0]]]], set[e[gp[x]]]]],
          not[member[w, binhom[NATADD, gp[x]]]], w → composite[t, PLUS]] // Reverse

Out[8]= or[equal[composite[t, PLUS], iterate[
            composite[gp[x], LEFT[APPLY[t, composite[id[omega], SUCC]]]], set[e[gp[x]]]]],
          not[member[composite[t, PLUS], binhom[NATADD, gp[x]]]] == True

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

Theorem. (The easy case.)

```
In[10]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
           not[implies[p1, p3]], {p1 → member[t, binhom[INTADD, gp[x]]],
            p2 -> member[composite[t, PLUS], binhom[NATADD, gp[x]]],
            p3 -> equal[composite[t, PLUS], iterate[composite[gp[x],
                LEFT[APPLY[t, composite[id[omega], SUCC]]]], set[e[gp[x]]]]]}]] // Reverse

Out[10]= or[equal[composite[t, PLUS], iterate[
             composite[gp[x], LEFT[APPLY[t, composite[id[omega], SUCC]]]], set[e[gp[x]]]]],
           not[member[t, binhom[INTADD, gp[x]]]] == True
```

```
In[11]:= or[equal[composite[t_, PLUS], iterate[
             composite[gp[x_], LEFT[APPLY[t_, composite[id[omega], SUCC]]]], set[e[gp[x_]]]]],
           not[member[t_, binhom[INTADD, gp[x_]]]]] := True
```

The other case requires only a little more work.

Lemma.

```
In[12]:= SubstTest[or,
           equal[w, iterate[composite[gp[x], LEFT[APPLY[w, set[0]]]], set[e[gp[x]]]]],
           not[member[w, binhom[NATADD, gp[x]]]], w → composite[t, INVERSE, PLUS]] // Reverse

Out[12]= or[equal[composite[t, INVERSE, PLUS], iterate[composite[gp[x],
              LEFT[APPLY[t, composite[inverse[SUCC], id[omega]]]], set[e[gp[x]]]]],
           not[member[composite[t, INVERSE, PLUS], binhom[NATADD, gp[x]]]] == True

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

Lemma.

```
In[14]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
             not[implies[p1, p3]], {p1 → member[t, binhom[INTADD, gp[x]]],
              p2 -> member[composite[t, INVERSE, PLUS], binhom[NATADD, gp[x]]],
              p3 -> equal[composite[t, INVERSE, PLUS], iterate[composite[gp[x], LEFT[
                   APPLY[t, composite[inverse[SUCC], id[omega]]]], set[e[gp[x]]]]]}]] // Reverse

Out[14]= or[equal[composite[t, INVERSE, PLUS],
             iterate[composite[gp[x], LEFT[APPLY[t, composite[inverse[SUCC], id[omega]]]]],
              set[e[gp[x]]]], not[member[t, binhom[INTADD, gp[x]]]] == True

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

One extra step is required: binary homomorphisms preserve inverses.

Lemma.

```
In[16]:= SubstTest[implies, member[t, binhom[gp[w], gp[x]]],
           equal[composite[t, inv[gp[w]]], composite[inv[gp[x]], t]], w → INTADD] // Reverse

Out[16]= or[equal[composite[inv[gp[x]], t], composite[t, id[Z], INVERSE]],
           not[member[t, binhom[INTADD, gp[x]]]] == True

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

The factor **id[Z]** is not needed here.

Theorem. A binary homomorphism from **INTADD** to a group **gp[x]** takes negatives to the inverses.

```
In[18]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p1, p3], implies[and[p2, p3], p4], not[implies[p1, p4]],
           {p1 -> member[t, binhom[INTADD, gp[x]]], p2 → equal[domain[t], Z],
            p3 -> equal[composite[inv[gp[x]], t], composite[t, id[Z], INVERSE]],
            p4 -> equal[composite[inv[gp[x]], t], composite[t, INVERSE]]}]] // Reverse

Out[18]= or[equal[composite[t, INVERSE], composite[inv[gp[x]], t]],
           not[member[t, binhom[INTADD, gp[x]]]]] == True

In[19]:= or[equal[composite[inv[gp[x_]], t_], composite[t_, INVERSE]],
           not[member[t_, binhom[INTADD, gp[x_]]]]] := True
```

In particular, the value of $\mathbf{t}$ at $-\mathbf{1}$ is the inverse of its value at $+\mathbf{1}$.

Lemma.

```
In[20]:= SubstTest[implies, equal[u, v],
           equal[APPLY[u, w], APPLY[v, w]], {u -> composite[funpart[t], INVERSE],
            v -> composite[inv[gp[x]], funpart[t]], w → plus[set[0]]}] // Reverse

Out[20]= or[equal[APPLY[funpart[t], composite[inverse[SUCC], id[omega]]],
            APPLY[inv[gp[x]], APPLY[funpart[t], composite[id[omega], SUCC]]]],
           not[equal[composite[funpart[t], INVERSE], composite[inv[gp[x]], funpart[t]]]]] == True

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

Lemma. (Remove the **funpart** wrapper.)

```
In[22]:= SubstTest[implies, equal[t, funpart[w]],
           or[equal[APPLY[t, composite[inverse[SUCC], id[omega]]],
             APPLY[inv[gp[x]], APPLY[t, composite[id[omega], SUCC]]]],
            not[equal[composite[t, INVERSE], composite[inv[gp[x]], t]]]], w → t] // Reverse

Out[22]= or[equal[APPLY[t, composite[inverse[SUCC], id[omega]]],
            APPLY[inv[gp[x]], APPLY[t, composite[id[omega], SUCC]]]],
           not[equal[composite[t, INVERSE], composite[inv[gp[x]], t]]], not[FUNCTION[t]]] == True

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

Finally, the **FUNCTION** literal can be omitted.

Theorem.

```
In[24]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
           not[implies[p1, p4]], {p1 → member[t, binhom[INTADD, gp[x]]], p2 → FUNCTION[t],
            p3 -> equal[composite[t, INVERSE], composite[inv[gp[x]], t]],
            p4 -> equal[APPLY[t, composite[inverse[SUCC], id[omega]]],
              APPLY[inv[gp[x]], APPLY[t, composite[id[omega], SUCC]]]]}]] // Reverse

Out[24]= or[equal[APPLY[t, composite[inverse[SUCC], id[omega]]],
            APPLY[inv[gp[x]], APPLY[t, composite[id[omega], SUCC]]]],
           not[member[t, binhom[INTADD, gp[x]]]]] == True
```

```
In[25]:= (% /. {t → t_, x → x_}) /. Equal → SetDelayed
```

Theorem. (The harder case.)

```
In[26]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p1, p3], implies[and[p2, p3], p4], not[implies[p1, p4]],
           {p1 -> member[t, binhom[INTADD, gp[x]]], p2 -> equal[composite[t, INVERSE, PLUS],
              iterate[composite[gp[x], LEFT[APPLY[t, composite[inverse[SUCC], id[omega]]]]],
               set[e[gp[x]]]]], p3 -> equal[APPLY[t, composite[inverse[SUCC], id[omega]]],
              APPLY[inv[gp[x]], APPLY[t, composite[id[omega], SUCC]]]], p4 -> equal[
              composite[t, INVERSE, PLUS], iterate[composite[gp[x], LEFT[APPLY[inv[gp[x]],
                  APPLY[t, composite[id[omega], SUCC]]]], set[e[gp[x]]]]]}]] // Reverse

Out[26]= or[equal[composite[inv[gp[x]], iterate[
             composite[gp[x], LEFT[APPLY[t, composite[id[omega], SUCC]]]], set[e[gp[x]]]],
           composite[t, INVERSE, PLUS]], not[member[t, binhom[INTADD, gp[x]]]] == True

In[27]:= or[equal[composite[inv[gp[x_]], iterate[
             composite[gp[x_], LEFT[APPLY[t_, composite[id[omega], SUCC]]]], set[e[gp[x_]]]],
           composite[t_, INVERSE, PLUS]], not[member[t_, binhom[INTADD, gp[x_]]]] := True
```

Lemma.

```
In[28]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
           not[implies[p1, p4]], {p1 -> member[t, binhom[INTADD, x]], p2 → FUNCTION[t],
            p3 → equal[domain[t], Z], p4 -> subclass[t, cart[Z, V]]}]] // Reverse

Out[28]= or[not[member[t, binhom[INTADD, x]]], subclass[t, cart[Z, V]]] == True

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

Main Theorem. Any binary homomorphism **t** from **INTADD** to a group **gp[x]** is determined by the element **APPLY[t, id[ω] ∘ SUCC]**.

```
In[30]:= (Map[not, SubstTest[and, implies[and[p1, p2, p3], p4], implies[and[p1, p2], p5], implies[
              and[p1, p3], p6], implies[and[p4, p5, p6], p7], not[implies[and[p1, p2, p3], p7]],
             {p1 → member[t, binhom[INTADD, gp[x]]], p2 → equal[u, composite[t, PLUS]],
              p3 → equal[v, composite[t, INVERSE, PLUS]], p4 → equal[t,
                union[composite[u, inverse[PLUS]], composite[v, inverse[PLUS], INVERSE]]], p5 →
               equal[u, iterate[composite[gp[x], LEFT[APPLY[t, composite[id[omega], SUCC]]]],
                 set[e[gp[x]]]]], p6 → equal[v, iterate[composite[gp[x], LEFT[
                   APPLY[inv[gp[x]], APPLY[t, composite[id[omega], SUCC]]]]], set[e[gp[x]]]],
              p7 → equal[t, union[composite[iterate[composite[gp[x], LEFT[
                     APPLY[t, composite[id[omega], SUCC]]]], set[e[gp[x]]]], inverse[PLUS]],
                 composite[iterate[composite[gp[x], LEFT[APPLY[inv[gp[x]], APPLY[t, composite[
                       id[omega], SUCC]]]]], set[e[gp[x]]]], inverse[PLUS], INVERSE]]}]] //
           Reverse) /. {u -> composite[t, PLUS], v -> composite[t, INVERSE,
            PLUS]}

Out[30]= or[equal[t, union[
            composite[iterate[composite[gp[x], LEFT[APPLY[t, composite[id[omega], SUCC]]]],
              set[e[gp[x]]]], inverse[PLUS]], composite[inv[gp[x]],
             iterate[composite[gp[x], LEFT[APPLY[t, composite[id[omega], SUCC]]]],
              set[e[gp[x]]]], inverse[PLUS], INVERSE]]],
          not[member[t, binhom[INTADD, gp[x]]]]] == True

In[31]:= or[equal[t_, union[
            composite[iterate[composite[gp[x_], LEFT[APPLY[t_, composite[id[omega], SUCC]]]],
              set[e[gp[x_]]]], inverse[PLUS]], composite[inv[gp[x_]],
             iterate[composite[gp[x_], LEFT[APPLY[t_, composite[id[omega], SUCC]]]],
              set[e[gp[x_]]]], inverse[PLUS], INVERSE]]],
          not[member[t_, binhom[INTADD, gp[x_]]]]] := True
```

---

a special case

Corollary. A special case.

```
In[36]:= (SubstTest[or, equal[t, union[
              composite[iterate[composite[gp[w], LEFT[APPLY[t, composite[id[omega], SUCC]]]],
                set[e[gp[w]]]], inverse[PLUS]], composite[inv[gp[w]],
               iterate[composite[gp[w], LEFT[APPLY[t, composite[id[omega], SUCC]]]],
                set[e[gp[w]]]], inverse[PLUS], INVERSE]]],
            not[member[t, binhom[INTADD, gp[w]]]], w → INTADD] //
           Reverse) /. t → inttimes[int[x]]

Out[36]= equal[inttimes[int[x]], union[composite[MIXMUL, LEFT[int[x]], inverse[PLUS]],
           composite[INVERSE, MIXMUL, LEFT[int[x]], inverse[PLUS], INVERSE]]] == True

In[38]:= union[composite[MIXMUL, LEFT[int[x_]], inverse[PLUS]], composite[INVERSE,
           MIXMUL, LEFT[int[x_]], inverse[PLUS], INVERSE]] := inttimes[int[x]]
```

Corollary.

```
In[48]:= SubstTest[implies, equal[x, int[t]],
           equal[union[composite[MIXMUL, LEFT[x], inverse[PLUS]], composite[INVERSE,
              MIXMUL, LEFT[x], inverse[PLUS], INVERSE]], inttimes[x]], t → x] // Reverse

Out[48]= or[equal[inttimes[x], union[composite[MIXMUL, LEFT[x], inverse[PLUS]], composite[
              INVERSE, MIXMUL, LEFT[x], inverse[PLUS], INVERSE]]], not[member[x, Z]]] == True

In[49]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[53]:= SubstTest[implies, and[empty[u], empty[v]], equal[u, v],
           {u → inttimes[x], v -> union[composite[MIXMUL, LEFT[x], inverse[PLUS]],
              composite[INVERSE, MIXMUL, LEFT[x], inverse[PLUS], INVERSE]]}] // Reverse

Out[53]= or[equal[inttimes[x], union[composite[MIXMUL, LEFT[x], inverse[PLUS]],
             composite[INVERSE, MIXMUL, LEFT[x], inverse[PLUS], INVERSE]]], member[x, Z]] == True

In[54]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[55]:= SubstTest[and, implies[p, q], or[p, q], {p → member[x, Z],
           q -> equal[inttimes[x], union[composite[MIXMUL, LEFT[x], inverse[PLUS]],
              composite[INVERSE, MIXMUL, LEFT[x], inverse[PLUS], INVERSE]]]}]

Out[55]= equal[inttimes[x], union[composite[MIXMUL, LEFT[x], inverse[PLUS]],
            composite[INVERSE, MIXMUL, LEFT[x], inverse[PLUS], INVERSE]]] == True

In[57]:= union[composite[MIXMUL, LEFT[x_], inverse[PLUS]],
           composite[INVERSE, MIXMUL, LEFT[x_], inverse[PLUS], INVERSE]] := inttimes[x]
```

reformulation

The general, but complicated, explicit formula derived in the first section has the following simple corollary.

Corollary.

```
In[58]:= Map[not, SubstTest[and, implies[p1, p3], implies[p1, p4],
           implies[and[p2, p3, p4], p5], not[implies[and[p1, p2], p5]],
           {p1 → and[member[u, binhom[INTADD, gp[x]]], member[v, binhom[INTADD, gp[x]]]],
            p2 → equal[APPLY[u, composite[id[omega], SUCC]],
              APPLY[v, composite[id[omega], SUCC]]], p3 → equal[u, union[
             composite[iterate[composite[gp[x], LEFT[APPLY[u, composite[id[omega], SUCC]]]],
               set[e[gp[x]]]], inverse[PLUS]], composite[iterate[composite[gp[x],
                LEFT[APPLY[inv[gp[x]], APPLY[u, composite[id[omega], SUCC]]]]],
               set[e[gp[x]]]], inverse[PLUS], INVERSE]]], p4 -> equal[v, union[
             composite[iterate[composite[gp[x], LEFT[APPLY[v, composite[id[omega], SUCC]]]],
               set[e[gp[x]]]], inverse[PLUS]], composite[iterate[composite[gp[x],
                LEFT[APPLY[inv[gp[x]], APPLY[v, composite[id[omega], SUCC]]]]],
               set[e[gp[x]]]], inverse[PLUS], INVERSE]]], p5 → equal[u, v]}]] // Reverse

Out[58]= or[equal[u, v], not[
            equal[APPLY[u, composite[id[omega], SUCC]], APPLY[v, composite[id[omega], SUCC]]],
           not[member[u, binhom[INTADD, gp[x]]]], not[member[v, binhom[INTADD, gp[x]]]]] == True

In[59]:= or[equal[u_, v_], not[equal[APPLY[u_, composite[id[omega], SUCC]],
             APPLY[v_, composite[id[omega], SUCC]]], not[member[u_, binhom[INTADD, gp[x_]]]],
           not[member[v_, binhom[INTADD, gp[x_]]]] := True
```

The variables **u** and **v** in this statement can be eliminated to obtain an elegant formulation of the fact that binary homomorphisms from integer addition to any group are uniquely determined by their values at **+ 1**.

Lemma.

```
In[60]:= member[pair[u, v], composite[inverse[eval[w]], eval[w]]] // AssertTest

Out[60]= member[pair[u, v], composite[inverse[eval[w]], eval[w]]] ==
          and[equal[APPLY[funpart[u], w], APPLY[funpart[v], w]],
           member[u, V], member[v, V], member[w, domain[funpart[u]]]]

In[61]:= member[pair[u_, v_], composite[inverse[eval[w_]], eval[w_]]] :=
          and[equal[APPLY[funpart[u], w], APPLY[funpart[v], w]],
           member[u, V], member[v, V], member[w, domain[funpart[u]]]]
```

Lemma.

```
In[62]:= SubstTest[implies, and[equal[u, funpart[s]], equal[v, funpart[t]]],
           or[equal[u, v], not[equal[APPLY[funpart[u], composite[id[omega], SUCC]], APPLY[
               funpart[v], composite[id[omega], SUCC]]]], not[member[u, binhom[INTADD, gp[x]]]],
            not[member[v, binhom[INTADD, gp[x]]]], {s → u, t → v}] // Reverse

Out[62]= or[equal[u, v], not[equal[APPLY[funpart[u], composite[id[omega], SUCC]],
             APPLY[funpart[v], composite[id[omega], SUCC]]],
           not[FUNCTION[u]], not[FUNCTION[v]], not[member[u, binhom[INTADD, gp[x]]]],
           not[member[v, binhom[INTADD, gp[x]]]] == True

In[63]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[64]:= Map[not, SubstTest[and, implies[p1, p3], implies[p1, p4],
           implies[and[p1, p2, p3, p4], p5], not[implies[and[p1, p2], p5]],
           {p1 → and[member[u, binhom[INTADD, gp[x]]], member[v, binhom[INTADD, gp[x]]]],
            p2 -> equal[APPLY[funpart[u], composite[id[omega], SUCC]],
              APPLY[funpart[v], composite[id[omega], SUCC]]],
            p3 → FUNCTION[u], p4 → FUNCTION[v], p5 → equal[u, v]}]] // Reverse

Out[64]= or[equal[u, v], not[equal[APPLY[funpart[u], composite[id[omega], SUCC]],
           APPLY[funpart[v], composite[id[omega], SUCC]]]],
         not[member[u, binhom[INTADD, gp[x]]]], not[member[v, binhom[INTADD, gp[x]]]]] == True

In[65]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Main Theorem. Binary homomorphisms from integer addition to any group are uniquely determined by their values at **+1**.

```
In[66]:= Map[empty[composite[Id, complement[#]]] &,
          SubstTest[class, pair[u, v], implies[member[pair[u, v], w], equal[u, v]],
           w -> composite[id[binhom[INTADD, gp[x]]], inverse[eval[composite[id[omega], SUCC]]],
            eval[composite[id[omega], SUCC]], id[binhom[INTADD, gp[x]]]]]]

Out[66]= FUNCTION[composite[id[binhom[INTADD, gp[x]]],
           inverse[eval[composite[id[omega], SUCC]]]] == True

In[67]:= FUNCTION[composite[id[binhom[INTADD, gp[x_]]],
           inverse[eval[composite[id[omega], SUCC]]]] := True
```

Corollary. A special case.

```
In[70]:= SubstTest[FUNCTION, composite[id[binhom[INTADD, gp[x]]],
           inverse[eval[composite[id[omega], SUCC]]], x → RATADD] // Reverse

Out[70]= FUNCTION[composite[id[binhom[INTADD, RATADD]],
           inverse[eval[composite[id[omega], SUCC]]]] == True

In[71]:= FUNCTION[composite[id[binhom[INTADD, RATADD]],
           inverse[eval[composite[id[omega], SUCC]]]] := True
```