binary homomorphisms from NATADD to NATMUL

Johan G. F. Belinfante
2013 January 3

```
In[1]:= SetDirectory["l:"]; << goedel.12dec27a

        :Package Title: goedel.12dec27a             2012 December 27 at 8:00 a.m.

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

        It is now:  2013 Jan 3 at 10:45

        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 Jan 3 at 11:1
```

summary

There is only one binary homomorphism from **NATADD** to **NATMUL** that fails to be a functor, namely $\omega \times \{0\}$. Every functor from **NATADD** to **NATMUL** has the form **NATEXP** $\circ$ **LEFT[x]** for some natural number **x**.

mapping properties

Theorem. Every binary homomorphism from one monoid to another is a mapping from the range of the one to the range of the other.

```
In[2]:= SubstTest[subclass, binhom[u, v],
          map[fix[domain[u]], fix[domain[v]]], {u → monoid[x], v → monoid[y]}] // Reverse

Out[2]= subclass[binhom[monoid[x], monoid[y]], map[range[monoid[x]], range[monoid[y]]]] == True

In[3]:= subclass[binhom[monoid[x_], monoid[y_]],
          map[range[monoid[x_]], range[monoid[y_]]]] := True
```

Corollary. Every binary homomorphism from **NATADD** to **NATMUL** is a mapping from ω to ω.

```
In[4]:= SubstTest[subclass, binhom[u, v],
          map[fix[domain[u]], fix[domain[v]]], {u → NATADD, v → NATMUL}] // Reverse

Out[4]= subclass[binhom[NATADD, NATMUL], map[omega, omega]] == True
```

```
In[5]:= subclass[binhom[NATADD, NATMUL], map[omega, omega]] := True
```

Corollary. Every functor from one monoid to another is a mapping from the range of the one to the range of the other.

```
In[6]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
          {p1 -> functor[t, monoid[x], monoid[y]], p2 → member[t, binhom[monoid[x], monoid[y]]],
           p3 -> member[t, map[range[monoid[x]], range[monoid[y]]]]}]] // Reverse

Out[6]= or[member[t, map[range[monoid[x]], range[monoid[y]]]],
          not[functor[t, monoid[x], monoid[y]]]] == True

In[7]:= or[member[t_, map[range[monoid[x_]], range[monoid[y_]]]],
          not[functor[t_, monoid[x_], monoid[y_]]]] := True
```

Corollary. Every functor from **NATADD** to **NATMUL** is a mapping from ω to ω.

```
In[8]:= SubstTest[implies, functor[t, monoid[x], monoid[y]],
          member[t, map[range[monoid[x]], range[monoid[y]]]], {x → NATADD, y → NATMUL}] // Reverse

Out[8]= or[member[t, map[omega, omega]], not[functor[t, NATADD, NATMUL]]] == True

In[9]:= or[member[t_, map[omega, omega]], not[functor[t_, NATADD, NATMUL]]] := True
```

an example

A binary homomorphism from **NATADD** to **NATMUL** need not be unital. Consequently, there is a binary homomorphism from **NATADD** to **NATMUL** that fails to be a functor.

Theorem. (An example.)

```
In[10]:= member[cart[omega, set[0]], binhom[NATADD, NATMUL]] // AssertTest

Out[10]= member[cart[omega, set[0]], binhom[NATADD, NATMUL]] == True

In[11]:= member[cart[omega, set[0]], binhom[NATADD, NATMUL]] := True
```

Theorem.

```
In[12]:= functor[cart[omega, set[0]], NATADD, NATMUL] // AssertTest

Out[12]= functor[cart[omega, set[0]], NATADD, NATMUL] == False

In[13]:= functor[cart[omega, set[0]], NATADD, NATMUL] := False
```

It will be shown below that this is the only binary homomorphism from **NATADD** to **NATMUL** that fails to be a functor.

Lemma. Binary homomorphisms preserve idempotents.

```
In[14]:= SubstTest[implies, member[funpart[t], binhom[x, y]],
           subclass[image[funpart[t], fix[composite[x, DUP]]], fix[composite[y, DUP]]],
           {x → NATADD, y → NATMUL}] // Reverse

Out[14]= or[not[member[0, domain[funpart[t]]]],
           not[member[funpart[t], binhom[NATADD, NATMUL]]],
           subclass[APPLY[funpart[t], 0], set[0]]] == True

In[15]:= (% /. t → t_) /. Equal → SetDelayed
```

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

```
In[16]:= SubstTest[implies, equal[t, funpart[w]],
            or[not[member[0, domain[t]]], not[member[t, binhom[NATADD, NATMUL]]],
             subclass[APPLY[t, 0], set[0]]], w → t] // MapNotNot // Reverse

Out[16]= or[not[member[0, domain[t]]],
           not[member[t, binhom[NATADD, NATMUL]]], subclass[APPLY[t, 0], set[0]]] == True

In[17]:= (% /. t → t_) /. Equal → SetDelayed
```

Theorem. Every binary homomorphism from **NATADD** to **NATMUL** takes **0** to either **0** or **1** = **{0}**.

```
In[18]:= Map[not, SubstTest[and, implies[p1, p2],
            implies[p2, p3], implies[and[p1, p3], p4], not[implies[p1, p4]],
            {p1 -> member[t, binhom[NATADD, NATMUL]], p2 → equal[domain[t], omega],
             p3 -> member[0, domain[t]], p4 -> subclass[APPLY[t, 0], set[0]]}]] // Reverse

Out[18]= or[not[member[t, binhom[NATADD, NATMUL]]], subclass[APPLY[t, 0], set[0]]] == True

In[19]:= or[not[member[t_, binhom[NATADD, NATMUL]]], subclass[APPLY[t_, 0], set[0]]] := True
```

Restatement.

```
In[20]:= implies[member[t, binhom[NATADD, NATMUL]],
          or[equal[APPLY[t, 0], 0], equal[APPLY[t, 0], set[0]]]]

Out[20]= True
```

functors from NATADD to NATMUL

Lemma. Every functor from **NATADD** to **NATMUL** can be written as an iterate expression.

```
In[21]:= SubstTest[implies, functor[t, NATADD, monoid[x]],
           equal[t, iterate[composite[monoid[x], LEFT[APPLY[t, set[0]]]], set[e[monoid[x]]]],
           x → NATMUL] // Reverse

Out[21]= or[equal[t, iterate[times[APPLY[t, set[0]]], set[set[0]]]],
           not[functor[t, NATADD, NATMUL]]] == True
```

```
In[22]:= or[equal[t_, iterate[times[APPLY[t_, set[0]]], set[set[0]]]],
           not[functor[t_, NATADD, NATMUL]]] := True
```

Lemma. If **t** is a functor from **NATADD** to **NATMUL**, then **t(1)** $\in \omega$.

```
In[23]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
            {p1 -> functor[t, NATADD, NATMUL], p2 → member[t, map[omega, omega]],
             p3 -> member[APPLY[t, set[0]], omega]}]] // Reverse

Out[23]= or[member[APPLY[t, set[0]], omega], not[functor[t, NATADD, NATMUL]]] == True

In[24]:= or[member[APPLY[t_, set[0]], omega], not[functor[t_, NATADD, NATMUL]]] := True
```

Lemma.

```
In[25]:= SubstTest[implies, equal[x, nat[t]],
           equal[iterate[times[x], set[set[0]]], composite[NATEXP, LEFT[x]]], t → x] // Reverse

Out[25]= or[equal[composite[NATEXP, LEFT[x]], iterate[times[x], set[set[0]]]],
           not[member[x, omega]]] == True

In[26]:= or[equal[composite[NATEXP, LEFT[x_]], iterate[times[x_], set[set[0]]]],
           not[member[x_, omega]]] := True
```

Main Theorem. Every functor **t** from **NATADD** to **NATMUL** has the form **NATEXP** ∘ **LEFT[t(1)]**.

```
In[27]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[p3, p4],
            implies[and[p2, p4], p5], not[implies[p1, p5]], {p1 → functor[t, NATADD, NATMUL],
             p2 → equal[t, iterate[times[APPLY[t, set[0]]], set[set[0]]]],
             p3 → member[APPLY[t, set[0]], omega], p4 → equal[iterate[times[APPLY[t, set[0]]],
                set[set[0]]], composite[NATEXP, LEFT[APPLY[t, set[0]]]]],
             p5 → equal[t, composite[NATEXP, LEFT[APPLY[t, set[0]]]]]}]] // Reverse

Out[27]= or[equal[t, composite[NATEXP, LEFT[APPLY[t, set[0]]]]],
           not[functor[t, NATADD, NATMUL]]] == True

In[28]:= or[equal[t_, composite[NATEXP, LEFT[APPLY[t_, set[0]]]]],
           not[functor[t_, NATADD, NATMUL]]] := True
```

an application

In this section it is shown that the main result of the previous section implies a multiplicative law of exponents. However, a better result can be derived directly.

Lemma.

```
In[31]:= functor[composite[NATEXP, LEFT[x]], NATADD, NATMUL] // AssertTest

Out[31]= functor[composite[NATEXP, LEFT[x]], NATADD, NATMUL] == member[x, omega]

In[32]:= functor[composite[NATEXP, LEFT[x_]], NATADD, NATMUL] := member[x, omega]
```

Lemma.

```
In[33]:= SubstTest[implies, and[functor[x, NATADD, NATMUL], functor[y, NATADD, NATADD]],
           functor[composite[x, y], NATADD, NATMUL],
           {x → composite[NATEXP, LEFT[nat[u]]], y → times[nat[v]]}] // Reverse

Out[33]= functor[composite[NATEXP, LEFT[nat[u]], times[nat[v]]], NATADD, NATMUL] == True

In[34]:= (% /. {u → u_, v → v_}) /. Equal → SetDelayed
```

A multiplicative law of exponents for powers can be derived from this.

```
In[35]:= SubstTest[implies, functor[t, NATADD, NATMUL],
           equal[t, composite[NATEXP, LEFT[APPLY[t, set[0]]]]],
           t → composite[NATEXP, LEFT[nat[u]], times[nat[v]]]] // Reverse

Out[35]= equal[composite[NATEXP, LEFT[natexp[nat[u], nat[v]]]],
           composite[NATEXP, LEFT[nat[u]], times[nat[v]]]] == True
```

The above result will not be kept because one can do better, eliminating the **nat** wrappers.

Theorem. A better rewrite rule without **nat** wrappers.

```
In[36]:= Assoc[composite[NATEXP, LEFT[u]], NATMUL, LEFT[v]]

Out[36]= composite[NATEXP, LEFT[u], times[v]] == composite[NATEXP, LEFT[natexp[u, v]]]

In[37]:= composite[NATEXP, LEFT[u_], times[v_]] := composite[NATEXP, LEFT[natexp[u, v]]]
```

Theorem. A variable-free rule.

```
In[38]:= Map[rotate[inverse[#]] &,
           SubstTest[reify, u, composite[x, LEFT[u], y], {x → NATEXP, y → NATMUL}]] // Reverse

Out[38]= composite[NATEXP, cross[NATEXP, Id], inverse[ASSOC]] ==
          composite[NATEXP, cross[Id, NATMUL]]

In[39]:= composite[NATEXP, cross[NATEXP, Id], inverse[ASSOC]] :=
          composite[NATEXP, cross[Id, NATMUL]]
```

binhoms

In this section the main theorem about functors from **NATADD** to **NATMUL** is restated in terms of binary homomorphisms.

Theorem. Every unital binary homomorphism is a functor.

```
In[40]:= SubstTest[implies, and[member[t, binhom[monoid[x], monoid[y]]],
           equal[APPLY[t, e[monoid[x]]], e[monoid[y]]]],
          functor[t, monoid[x], monoid[y]], {x → NATADD, y → NATMUL}] // Reverse

Out[40]= or[functor[t, NATADD, NATMUL], not[equal[APPLY[t, 0], set[0]]],
          not[member[t, binhom[NATADD, NATMUL]]]] == True

In[41]:= or[functor[t_, NATADD, NATMUL], not[equal[APPLY[t_, 0], set[0]]],
          not[member[t_, binhom[NATADD, NATMUL]]]] := True
```

Lemma.

```
In[42]:= SubstTest[implies, equal[u, v],
          equal[composite[u, w], composite[v, w]], {u -> composite[funpart[t], NATADD],
           v -> composite[NATMUL, cross[funpart[t], funpart[t]]], w → LEFT[0]}] // Reverse

Out[42]= or[equal[composite[funpart[t], id[omega]],
           composite[times[APPLY[funpart[t], 0]], funpart[t]]],
          not[equal[composite[NATMUL, cross[funpart[t], funpart[t]]],
            composite[funpart[t], NATADD]]]] == True

In[43]:= (% /. t → t_) /. Equal → SetDelayed
```

Lemma.

```
In[44]:= Map[not, SubstTest[and, implies[p1, p3], implies[p3, p4], implies[and[p2, p4], p5],
           not[implies[and[p1, p2], p5]], {p1 → member[funpart[t], binhom[NATADD, NATMUL]],
            p2 → equal[APPLY[funpart[t], 0], 0], p3 → equal[composite[funpart[t], NATADD],
              composite[NATMUL, cross[funpart[t], funpart[t]]]], p4 ->
             equal[composite[funpart[t], id[omega]], composite[times[APPLY[funpart[t], 0]],
               funpart[t]]], p5 → equal[composite[funpart[t], id[omega]],
              cart[image[inverse[funpart[t]], omega], set[0]]]}]] // Reverse

Out[44]= or[equal[cart[image[inverse[funpart[t]], omega], set[0]],
           composite[funpart[t], id[omega]]], not[equal[0, APPLY[funpart[t], 0]]],
          not[member[funpart[t], binhom[NATADD, NATMUL]]]] == True

In[45]:= (% /. t → t_) /. Equal → SetDelayed
```

Lemma.

```
In[46]:= Map[not, SubstTest[and, implies[and[p1, p2], p5], implies[p1, p6],
           implies[p1, p7], implies[p7, p8], implies[and[p5, p6, p8], p9],
           not[implies[and[p1, p2], p9]], {p1 → member[funpart[t], binhom[NATADD, NATMUL]],
            p2 → equal[APPLY[funpart[t], 0], 0], p5 → equal[composite[funpart[t], id[omega]],
              cart[image[inverse[funpart[t]], omega], set[0]]],
            p6 → equal[domain[funpart[t]], omega], p7 → member[funpart[t], map[omega, omega]],
            p8 → equal[image[inverse[funpart[t]], omega], omega],
            p9 → equal[funpart[t], cart[omega, set[0]]]}]] // Reverse

Out[46]= or[equal[cart[omega, set[0]], funpart[t]], not[equal[0, APPLY[funpart[t], 0]]],
          not[member[funpart[t], binhom[NATADD, NATMUL]]]] == True
```

```
In[47]:= (% /. t → t_) /. Equal → SetDelayed
```

The final step is to remove the **funpart** wrapper.

Theorem. If **APPLY[t, 0] = 0**, then **t = ω × {0}**.

```
In[48]:= SubstTest[implies, equal[t, funpart[w]],
           or[equal[cart[omega, set[0]], t], not[equal[0, APPLY[t, 0]]],
            not[member[t, binhom[NATADD, NATMUL]]]], w → t] // Reverse // MapNotNot

Out[48]= or[equal[t, cart[omega, set[0]]],
           not[equal[0, APPLY[t, 0]]], not[member[t, binhom[NATADD, NATMUL]]]] == True

In[49]:= or[equal[t_, cart[omega, set[0]]], not[equal[0, APPLY[t_, 0]]],
           not[member[t_, binhom[NATADD, NATMUL]]]] := True
```

Theorem. Every non-zero binary homomorphism from **NATADD** to **NATMUL** is a functor.

```
In[50]:= Map[not, SubstTest[and, implies[p1, or[p2, p3]],
           implies[and[p1, p2], p4], implies[and[p1, p3], p5],
           not[implies[p1, or[p4, p5]]], {p1 → member[t, binhom[NATADD, NATMUL]],
            p2 → equal[APPLY[t, 0], 0], p3 -> equal[APPLY[t, 0], set[0]],
            p4 → equal[t, cart[omega, set[0]]], p5 → functor[t, NATADD, NATMUL]}]] // Reverse

Out[50]= or[equal[t, cart[omega, set[0]]],
           functor[t, NATADD, NATMUL], not[member[t, binhom[NATADD, NATMUL]]]] == True

In[51]:= or[equal[t_, cart[omega, set[0]]], functor[t_, NATADD, NATMUL],
           not[member[t_, binhom[NATADD, NATMUL]]]] := True
```

Corollary. Every non-zero binary homomorphism from **NATADD** to **NATMUL** is an exponential function.

```
In[52]:= Map[not, SubstTest[and, implies[p1, or[p2, p3]], implies[p3, p4],
           not[implies[p1, or[p2, p4]]], {p1 → member[t, binhom[NATADD, NATMUL]],
            p2 -> equal[t, cart[omega, set[0]]], p3 -> functor[t, NATADD, NATMUL],
            p4 → equal[t, composite[NATEXP, LEFT[APPLY[t, set[0]]]]]}]] // Reverse

Out[52]= or[equal[t, cart[omega, set[0]]], equal[t, composite[NATEXP, LEFT[APPLY[t, set[0]]]]],
           not[member[t, binhom[NATADD, NATMUL]]]] == True

In[53]:= or[equal[t_, cart[omega, set[0]]],
           equal[t_, composite[NATEXP, LEFT[APPLY[t_, set[0]]]]],
           not[member[t_, binhom[NATADD, NATMUL]]]] := True
```