functors from NATADD to a monoid

Johan G. F. Belinfante
2012 December 20

```
In[1]:= SetDirectory["l:"]; << goedel.12dec18a
```

```
:Package Title: goedel.12dec18a              2012 December 18 at 8:35 p.m.

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

It is now:  2012 Dec 20 at 7:49

Loading Simplification Rules

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

weightlimit = 40

Loading completed.

It is now:  2012 Dec 20 at 8:5
```

summary

For any element **y ∈ range[x]** of a monoid **x**, the **power sequence {e[x], y, y · y, ⋯ }** is given by **iterate[x ∘ LEFT[y], {e[x]}]**. The power sequence is a functor from **NATADD** to **x**.

```
In[2]:= implies[and[member[x, MONOIDS], member[y, range[x]]],
          functor[iterate[composite[x, LEFT[y]], set[e[x]]], NATADD, x]]
```

```
Out[2]= True
```

In this notebook it is shown that, conversely, any functor from **NATADD** to a monoid **x** is the power sequence of its value at **1 = {0}**.

use of the monoid wrapper

Results about monoids can also be restated using the **monoid[x]** wrapper.

Lemma.

```
In[8]:= SubstTest[implies, and[member[t, MONOIDS], member[y, range[t]]],
          functor[iterate[composite[t, LEFT[y]], set[e[t]]], NATADD, t], t → monoid[x]] // Reverse

Out[8]= or[equal[0, monoid[x]],
          functor[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
           NATADD, monoid[x]], not[member[y, range[monoid[x]]]]] == True

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

Theorem. Power sequences of monoid elements are functors.

```
In[11]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3], not[implies[p1, p3]],
            {p1 -> member[y, range[monoid[x]]], p2 → not[empty[monoid[x]]],
             p3 -> functor[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
               NATADD, monoid[x]]}]] // Reverse

Out[11]= or[functor[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
            NATADD, monoid[x]], not[member[y, range[monoid[x]]]]] == True

In[12]:= or[functor[iterate[composite[monoid[x_], LEFT[y_]], set[e[monoid[x_]]]],
            NATADD, monoid[x_]], not[member[y_, range[monoid[x_]]]]] := True
```

basic properties of functors for monoids

Theorem. Functors from **NATADD** to a monoid preserve the neutral element.

```
In[15]:= SubstTest[implies, and[member[w, MONOIDS], member[x, MONOIDS], functor[t, w, x]],
           equal[APPLY[t, e[w]], e[x]], w → NATADD] // Reverse

Out[15]= or[equal[APPLY[t, 0], e[x]], not[functor[t, NATADD, x]], not[member[x, MONOIDS]]] == True

In[16]:= or[equal[APPLY[t_, 0], e[x_]],
           not[functor[t_, NATADD, x_]], not[member[x_, MONOIDS]]] := True
```

Corollary.

```
In[20]:= SubstTest[implies, and[member[w, MONOIDS], functor[t, NATADD, w]],
           equal[APPLY[t, 0], e[w]], w → monoid[x]] // Reverse

Out[20]= or[equal[0, monoid[x]], equal[APPLY[t, 0], e[monoid[x]]],
           not[functor[t, NATADD, monoid[x]]]] == True

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

This corollary has a redundant literal that can be removed.

Theorem. Functors from **NATADD** to monoids preserve the neutral element.

```
In[23]:= SubstTest[and, implies[p, q], or[p, q], {p -> equal[0, monoid[x]],
           q -> or[equal[APPLY[t, 0], e[monoid[x]]], not[functor[t, NATADD, monoid[x]]]]}]

Out[23]= or[equal[APPLY[t, 0], e[monoid[x]]], not[functor[t, NATADD, monoid[x]]]] == True

In[25]:= or[equal[APPLY[t_, 0], e[monoid[x_]]], not[functor[t_, NATADD, monoid[x_]]]] := True
```

In general, functors only satisfy the following inclusion.

```
In[26]:= implies[functor[t, x, y], subclass[composite[t, x], composite[y, cross[t, t]]]]

Out[26]= True
```

This inclusion can be strengthened to an equation if **x** and **y** are monoids.

Theorem. An equation satisfied by functors between monoids.

```
In[27]:= SubstTest[implies,
           and[functor[t, u, v], category[v], equal[domain[u], cartsq[range[u]]]],
           equal[composite[t, u], composite[v, cross[t, t]]],
           {u → monoid[x], v → monoid[y]}] // Reverse

Out[27]= or[equal[composite[t, monoid[x]], composite[monoid[y], cross[t, t]]],
           not[functor[t, monoid[x], monoid[y]]]] == True

In[28]:= or[equal[composite[t_, monoid[x_]], composite[monoid[y_], cross[t_, t_]]],
           not[functor[t_, monoid[x_], monoid[y_]]]] := True
```

Corollary. (Restatement without the **monoid** wrappers.)

```
In[32]:= SubstTest[implies, and[equal[x, monoid[u]], equal[y, monoid[v]]],
            or[equal[composite[t, x], composite[y, cross[t, t]]], not[functor[t, x, y]]],
            {u → x, v → y}] // Reverse // MapNotNot

Out[32]= or[equal[composite[t, x], composite[y, cross[t, t]]],
           not[functor[t, x, y]], not[member[x, MONOIDS]], not[member[y, MONOIDS]]] == True

In[34]:= or[equal[composite[t_, x_], composite[y_, cross[t_, t_]]], not[functor[t_, x_, y_]],
           not[member[x_, MONOIDS]], not[member[y_, MONOIDS]]] := True
```

In particular, these results hold for functors from **NATADD** to a monoid.

Theorem.

```
In[36]:= SubstTest[implies, functor[t, monoid[w], monoid[x]], equal[
            composite[t, monoid[w]], composite[monoid[x], cross[t, t]]], w → NATADD] // Reverse

Out[36]= or[equal[composite[t, NATADD], composite[monoid[x], cross[t, t]]],
           not[functor[t, NATADD, monoid[x]]]] == True

In[37]:= or[equal[composite[t_, NATADD], composite[monoid[x_], cross[t_, t_]]],
           not[functor[t_, NATADD, monoid[x_]]]] := True
```

Theorem.

```
In[39]:= SubstTest[implies, and[functor[t, w, x], member[w, MONOIDS], member[x, MONOIDS]],
           equal[composite[t, w], composite[x, cross[t, t]]], w → NATADD] // Reverse

Out[39]= or[equal[composite[t, NATADD], composite[x, cross[t, t]]],
           not[functor[t, NATADD, x]], not[member[x, MONOIDS]]] == True

In[40]:= or[equal[composite[t_, NATADD], composite[x_, cross[t_, t_]]],
           not[functor[t_, NATADD, x_]], not[member[x_, MONOIDS]]] := True
```

---

derivation of the converse

Lemma.

```
In[43]:= SubstTest[implies, equal[u, v],
           equal[composite[u, w], composite[v, w]], {u -> composite[funpart[t], NATADD],
            v -> composite[x, cross[funpart[t], funpart[t]]], w → LEFT[set[0]]}] // Reverse

Out[43]= or[equal[composite[x, LEFT[APPLY[funpart[t], set[0]]], funpart[t]],
            composite[funpart[t], id[omega], SUCC]],
           not[equal[composite[x, cross[funpart[t], funpart[t]]],
             composite[funpart[t], NATADD]]]] == True

In[44]:= (% /. {x → x_, t → t_}) /. Equal → SetDelayed
```

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

```
In[45]:= SubstTest[implies, equal[t, funpart[w]],
           or[equal[composite[x, LEFT[APPLY[t, set[0]]], t], composite[t, id[omega], SUCC]],
            not[equal[composite[x, cross[t, t]], composite[t, NATADD]]]], w → t] // Reverse

Out[45]= or[equal[composite[t, id[omega], SUCC], composite[x, LEFT[APPLY[t, set[0]]], t]],
           not[equal[composite[t, NATADD], composite[x, cross[t, t]]]], not[FUNCTION[t]]] == True

In[46]:= (% /. {x → x_, t → t_}) /. Equal → SetDelayed
```

Lemma.

```
In[47]:= SubstTest[implies, functor[t, u, v],
           equal[domain[t], range[u]], {u → NATADD, v → x}] // Reverse

Out[47]= or[equal[omega, domain[t]], not[functor[t, NATADD, x]]] == True

In[48]:= or[equal[omega, domain[t_]], not[functor[t_, NATADD, x_]]] := True
```

Lemma. Iteration rule for binary homomorphisms from **NATADD** to a monoid.

```
In[49]:= Map[not, SubstTest[and, (*implies[p1,p2],implies[p1,p3],*)
            implies[and[p2, p3], p4], implies[p1, p5], (*implies[and[p2,p4,p5],p6],*)
            not[implies[p1, p6]], {p1 → functor[t, NATADD, monoid[x]], p2 → FUNCTION[t],
             p3 -> equal[composite[t, NATADD], composite[monoid[x], cross[t, t]]], p4 -> equal[
               composite[t, id[omega], SUCC], composite[monoid[x], LEFT[APPLY[t, set[0]]], t]],
             p5 → equal[domain[t], omega], p6 -> equal[composite[t, SUCC],
               composite[monoid[x], LEFT[APPLY[t, set[0]]], t]]}]] // Reverse

Out[49]= or[equal[composite[t, SUCC], composite[monoid[x], LEFT[APPLY[t, set[0]]], t]],
           not[functor[t, NATADD, monoid[x]]]] == True

In[50]:= or[equal[composite[t_, SUCC], composite[monoid[x_], LEFT[APPLY[t_, set[0]]], t_]],
           not[functor[t_, NATADD, monoid[x_]]]] := True
```

Corollary.

```
In[51]:= SubstTest[implies, equal[x, monoid[w]],
            or[equal[composite[t, SUCC], composite[x, LEFT[APPLY[t, set[0]]], t]],
             not[functor[t, NATADD, x]]], w → x] // Reverse // MapNotNot

Out[51]= or[equal[composite[t, SUCC], composite[x, LEFT[APPLY[t, set[0]]], t]],
           not[functor[t, NATADD, x]], not[member[x, MONOIDS]]] == True

In[52]:= or[equal[composite[t_, SUCC], composite[x_, LEFT[APPLY[t_, set[0]]], t_]],
           not[functor[t_, NATADD, x_]], not[member[x_, MONOIDS]]] := True
```

Lemma. The vertical section of a binary homomorphism at $\mathbf{0}$.

```
In[53]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
            implies[p1, p4], implies[and[p3, p4], p5], not[implies[and[p1, p2], p5]],
            {p1 → functor[t, NATADD, x], p2 → member[x, MONOIDS], p3 → equal[APPLY[t, 0], e[x]],
             p4 → FUNCTION[t], p5 → equal[image[t, set[0]], set[e[x]]]}]] // Reverse

Out[53]= or[equal[image[t, set[0]], set[e[x]]],
           not[functor[t, NATADD, x]], not[member[x, MONOIDS]]] == True

In[54]:= (% /. {t → t_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[55]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[and[p1, p2], p4],
            implies[and[p3, p4], p5], not[implies[and[p1, p2], p5]], {p1 → functor[t, NATADD, x],
             p2 → member[x, MONOIDS], p3 → equal[image[t, set[0]], set[e[x]]],
             p4 -> equal[composite[t, SUCC], composite[x, LEFT[APPLY[t, set[0]]], t]],
             p5 → equal[composite[t, id[omega]],
               iterate[composite[x, LEFT[APPLY[t, set[0]]]], set[e[x]]]]}]] // Reverse

Out[55]= or[equal[composite[t, id[omega]],
            iterate[composite[x, LEFT[APPLY[t, set[0]]]], set[e[x]]]],
           not[functor[t, NATADD, x]], not[member[x, MONOIDS]]] == True

In[56]:= (% /. {t → t_, x → x_}) /. Equal → SetDelayed
```

The proof of the main theorem uses the uniqueness theorem for **iterate**. Some steps of the proof can be omitted, as indicated by **(* ⋯ *)**.

Main Theorem. A functor from **NATADD** to a monoid **x** is the power sequence of the element **APPLY[t, {0}] ∈ range[x]**.

```
In[57]:= Map[not, SubstTest[and, (*implies[p1,p5],implies[p1,p6],implies[and[p1,p2],p7],*)
           implies[and[p5, p6, p7], p8], not[implies[and[p1, p2], p8]],
           {p1 → functor[t, NATADD, x], p2 → member[x, MONOIDS], p5 → equal[domain[t], omega],
            p6 → FUNCTION[t], p7 → equal[composite[t, id[omega]],
              iterate[composite[x, LEFT[APPLY[t, set[0]]]], set[e[x]]]], p8 ->
             equal[t, iterate[composite[x, LEFT[APPLY[t, set[0]]]], set[e[x]]]]}]] // Reverse

Out[57]= or[equal[t, iterate[composite[x, LEFT[APPLY[t, set[0]]]], set[e[x]]]],
           not[functor[t, NATADD, x]], not[member[x, MONOIDS]]] == True

In[58]:= or[equal[t_, iterate[composite[x_, LEFT[APPLY[t_, set[0]]]], set[e[x_]]]],
           not[functor[t_, NATADD, x_]], not[member[x_, MONOIDS]]] := True
```

Corollary.

```
In[63]:= SubstTest[or, equal[t, iterate[composite[w, LEFT[APPLY[t, set[0]]]], set[e[w]]]],
           not[functor[t, NATADD, w]], not[member[w, MONOIDS]], w → monoid[x]] // Reverse

Out[63]= or[equal[0, monoid[x]],
           equal[t, iterate[composite[monoid[x], LEFT[APPLY[t, set[0]]]], set[e[monoid[x]]]]],
           not[functor[t, NATADD, monoid[x]]]] == True

In[64]:= (% /. {t → t_, x → x_}) /. Equal → SetDelayed
```

A redundant literal can be eliminated here.

Theorem.

```
In[65]:= SubstTest[and, implies[p, q], or[p, q], {p -> equal[0, monoid[x]],
           q -> or[equal[t, iterate[composite[monoid[x], LEFT[APPLY[t, set[0]]]],
               set[e[monoid[x]]]]], not[functor[t, NATADD, monoid[x]]]]}]

Out[65]= or[equal[t, iterate[composite[monoid[x], LEFT[APPLY[t, set[0]]]], set[e[monoid[x]]]]],
           not[functor[t, NATADD, monoid[x]]]] == True

In[66]:= or[equal[t_, iterate[composite[monoid[x_], LEFT[APPLY[t_, set[0]]]],
             set[e[monoid[x_]]]]], not[functor[t, NATADD, monoid[x_]]]] := True
```