functors from NATADD to a monoid

Johan G. F. Belinfante
2013 January 24

In[1]:= **SetDirectory["l:"]; << goedel.13jan23a**

```
:Package Title: goedel.13jan23a              2013 January 24 at 6:10 a.m.

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

It is now:  2013 Jan 24 at 17:59

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 Jan 24 at 18:15
```

summary

Each functor **t** from **NATADD** to a monoid **monoid[x]** is the list of powers of the element **APPLY[t, {0}] ∈ range[monoid[x]]**. A corollary of this fact is derived.

derivation

There is an explicit formula for functors from **NATADD** to any monoid.

In[2]:= **implies[functor[t, NATADD, monoid[x]],
equal[t, iterate[composite[monoid[x], LEFT[APPLY[t, set[0]]]], set[e[monoid[x]]]]]]**

Out[2]= True

Comments. The set **monoid[x]** in general could be either empty or a monoid. In the present situation, however, the assumption **functor[t, NATADD, monoid[x]]** implies that **monoid[x]** is not empty. This explicit formula implies the following corollary.

Theorem.

```
In[3]:= Map[not, SubstTest[and, implies[p1, p3], implies[p1, p4],
          implies[and[p2, p3, p4], p5], not[implies[and[p1, p2], p5]],
          {p1 → and[functor[u, NATADD, monoid[x]], functor[v, NATADD, monoid[x]]],
           p2 → equal[APPLY[u, set[0]], APPLY[v, set[0]]], p3 → equal[u,
             iterate[composite[monoid[x], LEFT[APPLY[u, set[0]]]], set[e[monoid[x]]]]],
           p4 -> equal[v, iterate[composite[monoid[x], LEFT[APPLY[v, set[0]]]],
              set[e[monoid[x]]]]], p5 → equal[u, v]}]] // Reverse

Out[3]= or[equal[u, v], not[equal[APPLY[u, set[0]], APPLY[v, set[0]]]],
          not[functor[u, NATADD, monoid[x]]], not[functor[v, NATADD, monoid[x]]]] == True

In[4]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

The variables **u** and **v** in this statement can be eliminated to obtain an elegant formulation of the fact that functors from natural addition to any monoid are uniquely determined by their values at **1 = {0}**.

Lemma.

```
In[5]:= member[pair[u, v], composite[inverse[eval[w]], eval[w]]] // AssertTest

Out[5]= 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[6]:= 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[7]:= SubstTest[implies, and[equal[u, funpart[s]], equal[v, funpart[t]]],
          or[equal[u, v], not[equal[APPLY[funpart[u], set[0]], APPLY[funpart[v], set[0]]]],
           not[functor[u, NATADD, monoid[x]]],
           not[functor[v, NATADD, monoid[x]]]], {s → u, t → v}] // Reverse

Out[7]= or[equal[u, v], not[equal[APPLY[funpart[u], set[0]], APPLY[funpart[v], set[0]]]],
          not[FUNCTION[u]], not[FUNCTION[v]], not[functor[u, NATADD, monoid[x]]],
          not[functor[v, NATADD, monoid[x]]]] == True

In[8]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[9]:= 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[functor[u, NATADD, monoid[x]], functor[v, NATADD, monoid[x]]],
           p2 -> equal[APPLY[funpart[u], set[0]], APPLY[funpart[v], set[0]]],
           p3 → FUNCTION[u], p4 → FUNCTION[v], p5 → equal[u, v]}]] // Reverse

Out[9]= or[equal[u, v], not[equal[APPLY[funpart[u], set[0]], APPLY[funpart[v], set[0]]]],
          not[functor[u, NATADD, monoid[x]]], not[functor[v, NATADD, monoid[x]]]] == True

In[10]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Main Theorem. Functors from natural addition to any monoid are uniquely determined by their values at **1 = {0}**.

```
In[11]:= Map[empty[composite[Id, complement[#]]] &,
          SubstTest[class, pair[u, v], implies[member[pair[setpart[u], setpart[v]], w],
            equal[setpart[u], setpart[v]]], w -> composite[id[func[NATADD, monoid[x]]],
             inverse[eval[set[0]]], eval[set[0]], id[func[NATADD, monoid[x]]]]]]

Out[11]= FUNCTION[composite[id[func[NATADD, monoid[x]]], inverse[eval[set[0]]]]] == True

In[12]:= FUNCTION[composite[id[func[NATADD, monoid[x_]]], inverse[eval[set[0]]]]] := True
```