monotonicity of addition

Johan G. F. Belinfante
2002 June 19

```
<< goedel52.o61; << tools.m
```

```
:Package Title: goedel52.o61         2002 June 18 at 10:50 p.m.
```

```
It is now:  2002 Jun 19 at 5:7
```

```
Loading Simplification Rules
```

```
TOOLS.M                  Revised 2002 June 12
```

```
weightlimit = 40
```

■ Introduction

A formula expressing the monotonicity of addition is derived in ths notebook for the case that one of the summands is fixed. The argument uses induction, and the relation **COMMUTE**. The basic commutativity result used is that if **x** commutes with **y** and **z**, then **x** also commutes with **composite[y,z]**. Because **NATADD** does not currently satisfy any normality tests, it needs to be sequestered from the action of assert in several steps.

■ some sethood lemmas

For convenience we introduce a temporary abbreviation:

```
r[x_] := restrict[x, omega, omega]
```

Since **omega** is a set, all such restrictions are sets:

```
member[r[x], V] // AssertTest
```

```
member[composite[id[omega], x, id[omega]], V] == True
```

```
member[composite[id[omega], x_, id[omega]], V] := True
```

In particular:

```
member[r[S], V]
```

```
True
```

Various expressions involving **NATADD** are also sets, but one needs to sequester **NATADD** from the action of assert to derive these:

```
(member[composite[funpart[z], RIGHT[x]], V] // AssertTest) /. z -> composite[SUCC, NATADD]

member[composite[SUCC, NATADD, RIGHT[x]], V] == True

member[composite[SUCC, NATADD, RIGHT[x_]], V] := True

(member[composite[funpart[z], r[S]], V] // AssertTest) /. z -> composite[NATADD, RIGHT[x]]

member[composite[NATADD, RIGHT[x], id[omega], S, id[omega]], V] == True

member[composite[NATADD, RIGHT[x_], id[omega], S, id[omega]], V] := True

(member[composite[funpart[z], r[S]], V] // AssertTest) /.
 z -> composite[SUCC, NATADD, RIGHT[x]]

member[composite[SUCC, NATADD, RIGHT[x], id[omega], S, id[omega]], V] == True

member[composite[SUCC, NATADD, RIGHT[x_], id[omega], S, id[omega]], V] := True
```

■ a membership rule

For convenience we introduce the following function:

```
lambda[x, composite[n, RIGHT[x]]] /. n -> NATADD

VERTSECT[composite[SWAP, inverse[rotate[NATADD]]]]
```

A temporary membership rule is needed to make good use of this function.

```
member[pair[u, v], composite[w, IMAGE[cross[x, y]]]] // AssertTest

member[pair[u, v], composite[w, IMAGE[cross[x, y]]]] ==
 and[member[u, V], member[v, V], member[composite[y, u, inverse[x]], V],
  member[pair[composite[y, u, inverse[x]], v], w]]

member[pair[u_, v_], composite[w_, IMAGE[cross[x_, y_]]]] :=
 and[member[u, V], member[v, V], member[composite[y, u, inverse[x]], V],
  member[pair[composite[y, u, inverse[x]], v], w]]
```

■ an induction argument

The induction step will be essentially this argument:

```
SubstTest[implies, and[commute[w, y], commute[w, z]], commute[w, composite[y, z]],
    {w -> r[S], y -> r[SUCC], z -> composite[NATADD, RIGHT[x_]]}]

or[equal[composite[id[omega], inverse[IMAGE[inverse[NATADD]]],
    inverse[IMAGE[id[cart[V, singleton[x_]]]]]], inverse[IMAGE[FIRST]], E],
   composite[SUCC, NATADD, RIGHT[x_], id[omega], S, id[omega]]],
  not[equal[composite[id[omega], S, NATADD, RIGHT[x_]],
    composite[NATADD, RIGHT[x_], id[omega], S, id[omega]]]]] == True
```

We add this as a temporary rule

```
or[equal[composite[id[omega], inverse[IMAGE[inverse[NATADD]]],
    inverse[IMAGE[id[cart[V, singleton[x_]]]]], inverse[IMAGE[FIRST]], E],
   composite[SUCC, NATADD, RIGHT[x_], id[omega], S, id[omega]]],
  not[equal[composite[id[omega], S, NATADD, RIGHT[x_]],
    composite[NATADD, RIGHT[x_], id[omega], S, id[omega]]]]] := True
```

The base case for the induction proof is not a problem:

```
member[0, image[inverse[VERTSECT[composite[SWAP, inverse[rotate[NATADD]]]]],
  image[COMMUTE, singleton[composite[id[omega], S, id[omega]]]]]]
```

```
True
```

Here is the induction step, rewritten:

```
implies[member[x, y], member[succ[x], y]] /.
 y -> image[inverse[VERTSECT[composite[SWAP, inverse[rotate[NATADD]]]]],
   image[COMMUTE, singleton[composite[id[omega], S, id[omega]]]]]
```

```
True
```

Our next step is to eliminate the variable **x** in the above statement:

```
Map[equal[V, #] &, SubstTest[class, x, implies[member[x, y], member[succ[x], y]],
   y -> image[inverse[VERTSECT[composite[SWAP, inverse[rotate[NATADD]]]]],
     image[COMMUTE, singleton[composite[id[omega], S, id[omega]]]]]]] // Reverse
```

```
subclass[image[SUCC, image[inverse[VERTSECT[composite[SWAP, inverse[rotate[NATADD]]]]],
    fix[composite[inverse[IMAGE[cross[Id, composite[id[omega], S, id[omega]]]]],
      IMAGE[cross[composite[id[omega], inverse[S], id[omega]], Id]]]]]],
  image[inverse[VERTSECT[composite[SWAP, inverse[rotate[NATADD]]]]],
   fix[composite[inverse[IMAGE[cross[Id, composite[id[omega], S, id[omega]]]]],
     IMAGE[cross[composite[id[omega], inverse[S], id[omega]], Id]]]]]] == True
```

We add this as a temporary rule:

```
subclass[image[SUCC, image[inverse[VERTSECT[composite[SWAP, inverse[rotate[NATADD]]]]],
    fix[composite[inverse[IMAGE[cross[Id, composite[id[omega], S, id[omega]]]]],
      IMAGE[cross[composite[id[omega], inverse[S], id[omega]], Id]]]]]],
  image[inverse[VERTSECT[composite[SWAP, inverse[rotate[NATADD]]]]],
   fix[composite[inverse[IMAGE[cross[Id, composite[id[omega], S, id[omega]]]]],
     IMAGE[cross[composite[id[omega], inverse[S], id[omega]], Id]]]]]] := True
```

The induction theorem can now be applied:

```
SubstTest[implies, and[member[0, w], subclass[image[SUCC, w], w]], subclass[omega, w],
    w -> image[inverse[VERTSECT[composite[SWAP, inverse[rotate[NATADD]]]]],
   image[COMMUTE, singleton[composite[id[omega], S, id[omega]]]]]]
```

```
subclass[omega, image[inverse[VERTSECT[composite[SWAP, inverse[rotate[NATADD]]]]],
   fix[composite[inverse[IMAGE[cross[Id, composite[id[omega], S, id[omega]]]]],
     IMAGE[cross[composite[id[omega], inverse[S], id[omega]], Id]]]]]] == True
```

```
subclass[omega, image[inverse[VERTSECT[composite[SWAP, inverse[rotate[NATADD]]]]],
   fix[composite[inverse[IMAGE[cross[Id, composite[id[omega], S, id[omega]]]]],
     IMAGE[cross[composite[id[omega], inverse[S], id[omega]], Id]]]]]] := True
```

The result looks better if one reintroduces a variable:

```
SubstTest[implies, and[member[x, y], subclass[y, z]], member[x, z],
    {y -> omega,
   z -> image[inverse[VERTSECT[composite[SWAP, inverse[rotate[NATADD]]]]],
     fix[composite[inverse[IMAGE[cross[Id, composite[id[omega], S, id[omega]]]]],
       IMAGE[cross[composite[id[omega], inverse[S], id[omega]], Id]]]]]}] // MapNotNot
```

```
or[equal[composite[id[omega], S, NATADD, RIGHT[x]],
   composite[NATADD, RIGHT[x], id[omega], S, id[omega]]], not[member[x, omega]]] == True
```

We add this as a temporary rule, but it wil be replaced later with a better rule.

```
or[equal[composite[id[omega], S, NATADD, RIGHT[x_]], composite[NATADD,
    RIGHT[x_], id[omega], S, id[omega]]], not[member[x_, omega]]] := True
```

■ final steps

The final steps are designed to remove the condition that **x** be a member of **omega**. The basic idea is that if this is not the case, then both sides of the equation reduce to **0**. We begin with a simplification rule:

```
subclass[omega, 0] // AssertTest
```

```
subclass[omega, 0] == False
```

```
subclass[omega, 0] := False
```

With this rule in place, we have:

```
equal[0, composite[NATADD, RIGHT[x]]]
```

```
not[member[x, omega]]
```

We now add a temporary rule:

```
SubstTest[implies, equal[y, z], equal[composite[u, y], composite[u, z]],
    {u -> r[S], y -> composite[NATADD, RIGHT[x]], z -> 0}]
```

```
or[member[x, omega],
  subclass[omega, complement[range[iterate[SUCC, singleton[x]]]]]] == True
```

```
or[member[x_, omega],
  subclass[omega, complement[range[iterate[SUCC, singleton[x_]]]]]] := True
```

The transitive property of equality is needed:

```
SubstTest[implies, and[equal[u, 0], equal[v, 0]], equal[u, v],
    {u -> composite[r[S], NATADD, RIGHT[x_]], v -> composite[NATADD, RIGHT[x_], r[S]]}]
```

```
or[equal[composite[id[omega], S, NATADD, RIGHT[x_]],
   composite[NATADD, RIGHT[x_], id[omega], S, id[omega]]], member[x_, omega],
  not[subclass[omega, complement[range[iterate[SUCC, singleton[x_]]]]]]] == True
```

```
or[equal[composite[id[omega], S, NATADD, RIGHT[x_]],
   composite[NATADD, RIGHT[x_], id[omega], S, id[omega]]], member[x_, omega],
  not[subclass[omega, complement[range[iterate[SUCC, singleton[x_]]]]]]] := True
```

We just need to do a bit of reasoning to complete the derivation

```
Map[not, SubstTest[and, implies[p1, p2],
  implies[not[p1], p3], implies[and[not[p1], p3], p2], not[p2],
    {p1 -> member[x, omega],
        p2 -> equal[composite[id[omega], S, NATADD, RIGHT[x]],
     composite[NATADD, RIGHT[x], id[omega], S, id[omega]]],
        p3 -> subclass[omega, complement[range[iterate[SUCC, singleton[x]]]]]}]]
```

```
equal[composite[id[omega], S, NATADD, RIGHT[x]],
  composite[NATADD, RIGHT[x], id[omega], S, id[omega]]] == True
```

It is not entirely clear how to orient this equation, but composites with functions behave better with respect to complementation when the function is on the right. For this reason, it seems desirable to orient the rewrite rule this way:

```
composite[NATADD, RIGHT[x_], id[omega], S, id[omega]] :=
 composite[id[omega], S, NATADD, RIGHT[x]]
```