commutative law of multiplication

Johan G. F. Belinfante
2002 August 28

```
<< goedel52.p17; << tools.m

:Package Title: goedel52.p17        2002 August 27 at 10:40 p.m.

It is now:  2002 Aug 28 at 18:9

Loading Simplification Rules

TOOLS.M                   Revised 2002 August 22

weightlimit = 40
```

■ summary

In this notebook a distributive law for multiplication is used to derive the commutative law for multiplication. The idea is this: at this stage, we already have a recursion relation for multiplication of natural numbers which followed directly from the definition, but it only works on one side:

```
natmul[x, succ[y]]

natadd[x, natmul[x, y]]

natmul[succ[x], y]

natmul[succ[x], y]
```

On the other hand, we have in the meantime used a theorem about powers of commuting factors to derive this distributive law:

```
natmul[natadd[x, y], z]

natadd[natmul[x, z], natmul[y, z]]
```

The idea is now to set **y = 1** in this distributive law to derive the missing other−siderecursion relation. Then left and right multiplication satisfy the same recursion relation and the uniqueness of iteration can be used to show they are equal.

■ a variant of the distributive law

The first step will be to rewrite the distributive law in another way:

```
symdif[composite[NATMUL, RIGHT[y], NATADD, RIGHT[x]],
    composite[NATADD, RIGHT[natmul[x, y]], NATMUL, RIGHT[y]]] // VSNormality
```

```
union[intersection[composite[NATADD, RIGHT[natmul[x, y]], NATMUL, RIGHT[y]],
   composite[complement[NATMUL], RIGHT[y], NATADD, RIGHT[x]]],
  intersection[composite[NATMUL, RIGHT[y], NATADD, RIGHT[x]],
   composite[complement[NATADD], RIGHT[natmul[x, y]], NATMUL, RIGHT[y]]]] == 0
```

```
union[intersection[composite[NATADD, RIGHT[natmul[x_, y_]], NATMUL, RIGHT[y_]],
   composite[complement[NATMUL], RIGHT[y_], NATADD, RIGHT[x_]]],
  intersection[composite[NATMUL, RIGHT[y_], NATADD, RIGHT[x_]],
   composite[complement[NATADD], RIGHT[natmul[x_, y_]], NATMUL, RIGHT[y_]]]] := 0
```

```
SubstTest[equal, 0, symdif[u, v],
    {u -> composite[NATMUL, RIGHT[y], NATADD, RIGHT[x]],
    v -> composite[NATADD, RIGHT[natmul[x, y]], NATMUL, RIGHT[y]]}]
```

```
True == equal[composite[NATADD, RIGHT[natmul[x, y]], NATMUL, RIGHT[y]],
  composite[NATMUL, RIGHT[y], NATADD, RIGHT[x]]]
```

We now replace **equal** by **Equal** and set **x = 1**.

```
Equal[composite[NATADD, RIGHT[natmul[x, y]], NATMUL, RIGHT[y]],
  composite[NATMUL, RIGHT[y], NATADD, RIGHT[x]]] /. x -> singleton[0]
```

```
composite[NATADD,
  RIGHT[union[y, complement[image[V, intersection[omega, singleton[y]]]]]],
  NATMUL, RIGHT[y]] == composite[NATMUL, RIGHT[y], id[omega], SUCC]
```

This mess needs to be cleaned up a little to get the desired recursion relation.

■ cleaning up the recursion relation

The **GOEDEL** program sometimes recognizes the truth of an assertion even when the corresponding rewrite rule is missing. Here is an example:

```
equal[composite[NATMUL, RIGHT[x], id[omega]], composite[NATMUL, RIGHT[x]]]
```

```
True
```

This justifies adding the corresponding rewrite rule:

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

■ more cleaning up

The messy **RIGHT** factor can be cleaned up using a combination of associativity and **VSNormality**.

```
RIGHT[union[x, complement[image[V, intersection[y, singleton[z]]]]]] // VSNormality
```

```
RIGHT[union[x, complement[image[V, intersection[y, singleton[z]]]]]] == composite[
  id[cart[V, intersection[image[V, intersection[y, singleton[z]]], singleton[x]]]],
  inverse[FIRST]]
```

```
RIGHT[union[x_, complement[image[V, intersection[y_, singleton[z_]]]]]] := composite[
  id[cart[V, intersection[image[V, intersection[y, singleton[z]]], singleton[x]]]],
  inverse[FIRST]]

Assoc[id[cart[V, y]], id[cart[V, singleton[x]]], inverse[FIRST]] // Reverse

composite[id[cart[V, intersection[y, singleton[x]]]], inverse[FIRST]] ==
 composite[RIGHT[x], id[image[V, intersection[y, singleton[x]]]]]

composite[id[cart[V, intersection[y_, singleton[x_]]]], inverse[FIRST]] :=
 composite[RIGHT[x], id[image[V, intersection[y, singleton[x]]]]]

Assoc[NATADD, id[cart[omega, V]], RIGHT[x]]

composite[NATADD, RIGHT[x], id[intersection[omega, image[V, singleton[x]]]]] ==
 composite[NATADD, RIGHT[x]]

composite[NATADD, RIGHT[x_], id[intersection[omega, image[V, singleton[x_]]]]] :=
 composite[NATADD, RIGHT[x]]
```

■ the resulting cleaned up recursion relation

We are now ready to repeat the process of setting **x = 1** that we did earlier, and this time we do not get a mess.

```
Equal[composite[NATADD, RIGHT[natmul[x, y]], NATMUL, RIGHT[y]],
  composite[NATMUL, RIGHT[y], NATADD, RIGHT[x]]] /. x -> singleton[0]

composite[NATADD, RIGHT[y], NATMUL, RIGHT[y]] == composite[NATMUL, RIGHT[y], SUCC]
```

We turn this around and make it a rewrite rule:

```
composite[NATMUL, RIGHT[x_], SUCC] := composite[NATADD, RIGHT[x], NATMUL, RIGHT[x]]
```

Finally, the uniqueness theorem for **iterate** is used to derive one version of the commutative law:

```
SubstTest[implies, equal[composite[u, w], composite[w, SUCC]],
    equal[composite[w, id[omega]], iterate[u, image[w, singleton[0]]]],
    {u -> composite[NATADD, RIGHT[x]], w -> composite[NATMUL, RIGHT[x]]}]

equal[composite[NATMUL, LEFT[x]], composite[NATMUL, RIGHT[x]]] == True
```

It is unclear how best to orient this rule, so we avoid the issue for now:

```
equal[composite[NATMUL, LEFT[x_]], composite[NATMUL, RIGHT[x_]]] := True
```

■ SWAP rule

Other versions of the commutative law can be derived. For example:

```
SubstTest[assert,
 forall[x, equal[composite[z, RIGHT[x]], composite[z, LEFT[x]]]], z → NATMUL]

True == equal[rotate[NATMUL], rotate[composite[NATMUL, SWAP]]]
```

We replace **equal** with **Equal** and rotate twice:

```
Map[rotate[rotate[#]] &,
  Equal[rotate[NATMUL], rotate[composite[NATMUL, SWAP]]]] // Reverse
```

```
composite[NATMUL, SWAP] == NATMUL
```

This certainly can be added as a rewrite rule

```
composite[NATMUL, SWAP] := NATMUL
```

■ yet another version

Yet another version is obtained:

```
Map[A, ImageComp[NATMUL, SWAP, cart[singleton[x], singleton[y]]]]
```

```
natmul[x, y] == natmul[y, x]
```

This cannot be directly made into a rewrite rule, but we could add it as a fact:

```
Map[equal[#, natmul[y, x]] &, %]
```

```
equal[natmul[x, y], natmul[y, x]] == True
```

```
equal[natmul[x_, y_], natmul[y_, x_]] := True
```

Better yet, we will declare **natmul** to be **Orderless**. Then some existing rewrite rules can safely be removed.