division of natural numbers

Johan G. F. Belinfante
2003 August 27

```
In[1]:= << goedel52.s81; << tools.m

        :Package Title: goedel52.s81       2003 August 23 at 9:05 a.m.

        It is now:  2003 Aug 27 at 10:5

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 August 9

        weightlimit = 40
```

summary

We have followed the conventional wisdom that multiplication of natural numbers should be defined first, and then defined division of natural numbers in terms of multiplication using rotation. In this approach, the family of left−multiplication functions **composite[NATMUL, LEFT[x]]** were found to be solutions of iteration problems. These iteration equations suggested a membership rule for **NATMUL** from which the major properties of multipication could be derived, but it led to a rather complicated formula for the function **NATMUL** itself, which could be an obstacle for automated reasoning using **Otter**. In this notebook it is shown that the inverse of the relation **rotate[NATMUL]** is itself the solution of an iteration problem, yielding a much simpler formula for **NATMUL**. This suggests that for automated reasoning it may be advantageous to turn the tables and define division of natural numbers first, and then to define multiplication in terms of division via rotation. The relation **rotate[NATMUL]** is not quite a function because division of **0** by **0** does not yield a unique result. A restriction of it is a function:

```
In[2]:= FUNCTION[composite[rotate[NATMUL], id[cart[V, complement[singleton[0]]]]]]

Out[2]= True
```

The iteration problem considered in this notebook is for the inverse of the unrestricted relation **rotate[NATMUL]**.

derivation

Lemma 1.

```
In[3]:= composite[inverse[RIGHT[x]], inverse[NATMUL]] // DoubleInverse

Out[3]= composite[inverse[RIGHT[x]], inverse[NATMUL]] ==
         composite[inverse[LEFT[x]], inverse[NATMUL]]

In[4]:= composite[inverse[RIGHT[x_]], inverse[NATMUL]] :=
         composite[inverse[LEFT[x]], inverse[NATMUL]]
```

Lemma 2.

```
In[5]:= image[inverse[rotate[NATMUL]], singleton[0]] // DoubleInverse

Out[5]= composite[inverse[LEFT[0]], inverse[NATMUL]] == cart[singleton[0], omega]

In[6]:= composite[inverse[LEFT[0]], inverse[NATMUL]] := cart[singleton[0], omega]
```

The goal now is to derive a recursion relation for **inverse[rotate[NATMUL]]**. First a recursion relation for left−multiplication is derived:

```
In[7]:= symdif[composite[NATMUL, LEFT[succ[x]]],
          composite[NATADD, cross[composite[NATMUL, LEFT[x]], Id], DUP]] // VSNormality

Out[7]= union[intersection[composite[NATMUL, LEFT[succ[x]]], composite[complement[NATADD],
            id[composite[inverse[LEFT[x]], inverse[NATMUL]]], inverse[SECOND]]],
          intersection[composite[complement[NATMUL], LEFT[succ[x]]], composite[NATADD,
            id[composite[inverse[LEFT[x]], inverse[NATMUL]]], inverse[SECOND]]]] == 0

In[8]:= (% /. x -> x_) /. Equal -> SetDelayed

In[9]:= SubstTest[equal, 0, symdif[u, v], {u -> composite[NATMUL, LEFT[succ[x]]],
          v -> composite[NATADD, cross[composite[NATMUL, LEFT[x]], Id], DUP]}]

Out[9]= True == equal[composite[NATMUL, LEFT[succ[x]]],
          composite[NATADD, id[composite[inverse[LEFT[x]], inverse[NATMUL]]], inverse[SECOND]]]

In[10]:= composite[NATMUL, LEFT[succ[x_]]] :=
          composite[NATADD, id[composite[inverse[LEFT[x]], inverse[NATMUL]]], inverse[SECOND]]
```

Corollary.

```
In[11]:= composite[inverse[LEFT[succ[x]]], inverse[NATMUL]] // DoubleInverse

Out[11]= composite[inverse[LEFT[succ[x]]], inverse[NATMUL]] ==
          composite[SECOND, id[composite[inverse[LEFT[x]], inverse[NATMUL]]], inverse[NATADD]]

In[12]:= composite[inverse[LEFT[succ[x_]]], inverse[NATMUL]] :=
          composite[SECOND, id[composite[inverse[LEFT[x]], inverse[NATMUL]]], inverse[NATADD]]
```

The main recursion relation is now within reach:

```
In[13]:= symdif[composite[inverse[rotate[NATMUL]], SUCC],
           composite[intersection[composite[inverse[FIRST], NATADD],
             composite[inverse[SECOND], SECOND]], inverse[rotate[NATMUL]]]] // VSNormality

Out[13]= union[
           intersection[complement[composite[intersection[composite[inverse[FIRST], NATADD],
               composite[inverse[SECOND], SECOND]], inverse[rotate[NATMUL]]]],
            composite[inverse[rotate[NATMUL]], SUCC]], intersection[
            composite[complement[inverse[rotate[NATMUL]]], SUCC],
            composite[intersection[composite[inverse[FIRST], NATADD],
              composite[inverse[SECOND], SECOND]], inverse[rotate[NATMUL]]]]] == 0

In[14]:= % /. Equal -> SetDelayed

In[15]:= SubstTest[equal, 0, symdif[u, v], {u -> composite[inverse[rotate[NATMUL]], SUCC],
           v -> composite[intersection[composite[inverse[FIRST], NATADD],
              composite[inverse[SECOND], SECOND]], inverse[rotate[NATMUL]]]}]

Out[15]= True == equal[composite[intersection[composite[inverse[FIRST], NATADD],
             composite[inverse[SECOND], SECOND]], inverse[rotate[NATMUL]]],
           composite[inverse[rotate[NATMUL]], SUCC]]
```

```
In[16]:= composite[inverse[rotate[NATMUL]], SUCC] :=
          composite[intersection[composite[inverse[FIRST], NATADD],
            composite[inverse[SECOND], SECOND]], inverse[rotate[NATMUL]]]
```

The uniqueness of iteration is used to obtain the main result:

```
In[17]:= SubstTest[implies, and[equal[image[w, singleton[0]], v],
                 equal[composite[w, SUCC], composite[u, w]]],
               equal[composite[w, id[omega]], iterate[u, v]],
               {u -> intersection[
               composite[inverse[FIRST], NATADD], composite[inverse[SECOND], SECOND]],
                   v -> cart[singleton[0], omega], w -> inverse[rotate[NATMUL]]}]


Out[17]= equal[inverse[rotate[NATMUL]],
           iterate[intersection[composite[inverse[FIRST], NATADD],
             composite[inverse[SECOND], SECOND]], cart[singleton[0], omega]]] == True

In[18]:= iterate[
           intersection[composite[inverse[FIRST], NATADD], composite[inverse[SECOND], SECOND]],
           cart[singleton[0], omega]] := inverse[rotate[NATMUL]]
```

The beauty of this result is that it gives a formula for **NATMUL** itself instead of for its composite with **LEFT[x]**.

comments

The idea that led to the formula derived here is the distributive law: left−multiplication by a natural number can be viewed as a homomorphism of addition. The sum of two homomorphisms is a homomorphism, so left−multiplication by a number **n** can be viewed as the sum of **n** copies of left−multiplication by 1, that is, as **n** copies of the function **id[omega]**. One can define these functions iteratively, starting with left−multiplication by **0**, which is the constant function **cart[omega, singleton[0]]**. The left−multiplication functions are vertical sections of a relation:

```
In[19]:= image[composite[SWAP, inverse[rotate[NATMUL]]], singleton[x]]

Out[19]= composite[NATMUL, LEFT[x]]
```

This approach yields as iteration equation for the inverse of the flip of **rotate[NATADD]**. The final derivation exhibited above did not require giving a precise definition of what is meant by a homomorphism. The sum of homomorphisms **f** and **g** is conventionally written as **h(x) = f(x) + g(x)**, which translates into the variable−free formula

```
In[20]:= h == composite[NATADD, cross[f, g], DUP]

Out[20]= h == composite[NATADD,
           intersection[composite[inverse[FIRST], f], composite[inverse[SECOND], g]]]
```

This expression can be simplified by replacing **g = id[omega]** by **Id**, and rewritten as an image of **f** by abstraction:

```
In[21]:= (abstract[f, composite[x, cross[f, Id], y]]) /. {x → NATADD, y → DUP}

Out[21]= composite[cross[inverse[DUP], NATADD],
          id[composite[inverse[SECOND], SECOND]], cross[inverse[FIRST], inverse[FIRST]]]
```

This expression can be rewritten in various ways, for example:

```
In[22]:= composite[cross[inverse[DUP], NATADD], id[composite[inverse[SECOND], SECOND]],
           cross[inverse[FIRST], inverse[FIRST]]] // twist // twist

Out[22]= twist[composite[SWAP, inverse[rotate[NATADD]], inverse[DUP]]]
```

The following seems particularly simple:

```
In[23]:= (twist[composite[SWAP, inverse[rotate[x]], inverse[DUP]]] // VSTriNormality) /.
          x → NATADD

Out[23]= twist[composite[SWAP, inverse[rotate[NATADD]], inverse[DUP]]] ==
          composite[intersection[composite[inverse[FIRST], SECOND],
            composite[inverse[SECOND], NATADD]], SWAP]
```

the flipped iteration

The flipped iteration equation alluded to in the comments above is derived in this section using techniques similar to those used previously.

```
In[24]:= symdif[
           intersection[composite[inverse[FIRST], FIRST], composite[inverse[SECOND], NATADD]],
             flip[intersection[composite[inverse[FIRST], SECOND],
             composite[inverse[SECOND], NATADD]]]] // VSNormality

Out[24]= intersection[composite[intersection[
             composite[inverse[FIRST], SECOND], composite[inverse[SECOND], NATADD]], SWAP],
           composite[inverse[SECOND], Di, NATADD]] == 0

In[25]:= intersection[composite[intersection[
             composite[inverse[FIRST], SECOND], composite[inverse[SECOND], NATADD]], SWAP],
           composite[inverse[SECOND], Di, NATADD]] := 0

In[26]:= SubstTest[equal, 0, symdif[u, v], {u -> intersection[
             composite[inverse[FIRST], FIRST], composite[inverse[SECOND], NATADD]],
             v -> flip[intersection[composite[inverse[FIRST], SECOND],
              composite[inverse[SECOND], NATADD]]]}]

Out[26]= True == equal[composite[intersection[
             composite[inverse[FIRST], SECOND], composite[inverse[SECOND], NATADD]], SWAP],
           intersection[composite[inverse[FIRST], FIRST], composite[inverse[SECOND], NATADD]]]

In[27]:= composite[intersection[composite[inverse[FIRST], SECOND],
             composite[inverse[SECOND], NATADD]], SWAP] :=
          intersection[composite[inverse[FIRST], FIRST], composite[inverse[SECOND], NATADD]]

In[28]:= Assoc[composite[intersection[composite[inverse[FIRST], SECOND],
             composite[inverse[SECOND], NATADD]]], SWAP, SWAP] // Reverse

Out[28]= composite[intersection[composite[inverse[FIRST], FIRST],
             composite[inverse[SECOND], NATADD]], SWAP] ==
          intersection[composite[inverse[FIRST], SECOND], composite[inverse[SECOND], NATADD]]

In[29]:= composite[intersection[composite[inverse[FIRST], FIRST],
             composite[inverse[SECOND], NATADD]], SWAP] :=
          intersection[composite[inverse[FIRST], SECOND], composite[inverse[SECOND], NATADD]]
```

```
In[30]:= SubstTest[implies, and[equal[image[w, singleton[0]], v],
                equal[composite[w, SUCC], composite[u, w]]],
            equal[composite[w, id[omega]], iterate[u, v]],
            {u -> intersection[
            composite[inverse[FIRST], FIRST], composite[inverse[SECOND], NATADD]],
                v -> cart[omega, singleton[0]], w -> composite[SWAP, inverse[rotate[NATMUL]]]}]

Out[30]= equal[composite[SWAP, inverse[rotate[NATMUL]]],
           iterate[intersection[composite[inverse[FIRST], FIRST],
             composite[inverse[SECOND], NATADD]], cart[omega, singleton[0]]]] == True

In[31]:= iterate[
           intersection[composite[inverse[FIRST], FIRST], composite[inverse[SECOND], NATADD]],
           cart[omega, singleton[0]]] := composite[SWAP, inverse[rotate[NATMUL]]]
```