definition of NATMUL and a doubling formula

Johan G. F. Belinfante
2002 July 9

```
<< goedel52.o93; << tools.m

:Package Title: goedel52.o93          2002 July 9 at 7:40 a.m.

It is now:  2002 Jul 9 at 9:11

Loading Simplification Rules

TOOLS.M                  Revised 2002 June 12

weightlimit = 40
```

■ Introduction

The **GOEDEL** program is not an automated reasoning program, but only a an auxiliary program created to discover definitions and statements of theorems that can be used by an automated reasoning program such as **Otter**. If one wishes to base the arithmetic of natural numbers on set theory, one needs to begin with the definitions of the basic arithmetic operations, addition and multiplication, in terms of sets. This notebook is a progress report on what has been achieved to date on this issue.

Classes are generally introduced in the **GOEDEL** program via membership rules. Theoretically, one can obtain a formula for the class by applying Goedel's algorithm, but the catch is that the result may be rather complicated. In the case of addition, an elegant formula was found in terms of power, but no compact formula has yet been found for multiplication.

In practice the definitions of the functions **NATADD** and **NATMUL** are not needed for automated reasoning except to establish that the class exists without appealing to Goedel's class existence metatheorem. Introduce these classes via membership rules would amount to adding new axioms to set theory. The alternative explored here is to use the complicated definition to avoid introducing new axioms, and then to immediately derive membership (or other) rules from this definition. The bulk of the theory of arithmetic could then be based on these derived rules, and one can effectively forget about the definition. This is the essence of the axiomatic method: free the theory from the details of the definition by setting up appropriate axioms, prove a uniqueness theorem and then relegate the definition to the proof of an existence theorem.

In the final section a derivation of a variable−freestatement of **2 x = x + x** is presented. This formula was actually one of the first theorems proved about multiplication.

■ formula for NATADD

There is a simple formula for the function **NATADD** corresponding to addition of natural numbers:

```
composite[id[omega], rotate[inverse[power[SUCC]]], SWAP]

NATADD
```

An extensive body of facts about **NATADD** has been derived (using hand−guidedproofs) in the **GOEDEL** program, and this theory is now ready for work using **Otter**. This is the subject of other notebooks and will not be repeated here.

■ proposed formula for NATMUL

The situation for multiplication is slightly less satisfactory. The best definition found so far is this:

```
muldefn :=
 Equal[NATMUL, composite[rotate[composite[complement[composite[complement[composite[
        rotate[composite[inverse[power[SUCC]], SWAP]], RIF, cross[SECOND,
         composite[SWAP, SECOND]], id[composite[id[cart[omega, V]], inverse[FIRST],
          SUCC, FIRST]], cross[inverse[E], Id]]], id[composite[inverse[E],
        IMAGE[id[complement[cart[singleton[0], Id]]]]]]], inverse[FIRST]]],
    inverse[IMAGE[cross[Id, inverse[LEFT[0]]]]], E]], id[cart[omega, V]]]]
```

Despite the apparent complexity of this definition, it is feasible as a starting point. The first step in using this definition is to derive from it a formula for the composite of **NATMUL** with **LEFT[x]**.

```
Map[composite[#, LEFT[x]] &, muldefn]

composite[id[image[V, intersection[omega, singleton[x]]]],
  iterate[iterate[SUCC, singleton[x]], singleton[0]]] ==
 composite[iterate[iterate[SUCC, singleton[x]], singleton[0]],
  id[image[V, intersection[omega, singleton[x]]]]]
```

These are equal because **id[image[V,z]]** commutes with every relation: the **GOEDEL** program recognizes the truth of this when asked:

```
% /. Equal -> equal

True
```

From the formula for **composite[NATMUL,LEFT[x]]**, it is easy to derive the membership rule:

```
member[x, NATMUL]

and[member[first[first[x]], omega], member[pair[second[first[x]], second[x]],
  iterate[iterate[SUCC, singleton[first[first[x]]]], singleton[0]]]]
```

In the **GOEDEL** program, this membership rule has been used as the starting point for the theory of multiplication. At this point only a few facts have been derived:

```
FUNCTION[NATMUL]

True

domain[NATMUL]

cart[omega, omega]

range[NATMUL]

omega
```

Multiplication examples:

```
member[pair[pair[0, x], 0], NATMUL]
```

```
member[x, omega]
```

```
member[pair[pair[singleton[0], x], x], NATMUL]
```

```
member[x, omega]
```

■ A formula for doubling.

The formula **2 x = x + x** will be proved as follows. First we derive a duplication formula for addition:

```
Assoc[NATADD, cross[SUCC, Id], cross[Id, SUCC]]
```

```
composite[NATADD, cross[SUCC, SUCC]] == composite[SUCC, SUCC, NATADD]
```

```
composite[NATADD, cross[SUCC, SUCC]] := composite[SUCC, SUCC, NATADD]
```

```
Assoc[NATADD, cross[SUCC, SUCC], DUP]
```

```
composite[NATADD, DUP, SUCC] == composite[SUCC, SUCC, NATADD, DUP]
```

```
composite[NATADD, DUP, SUCC] := composite[SUCC, SUCC, NATADD, DUP]
```

```
ImageComp[NATADD, RIGHT[0], singleton[0]] // Reverse
```

```
image[NATADD, cart[singleton[0], singleton[0]]] == singleton[0]
```

```
image[NATADD, cart[singleton[0], singleton[0]]] := singleton[0]
```

The uniqueness of **iterate** implies:

```
SubstTest[implies, and[equal[image[w, singleton[0]], v],
        equal[composite[u, w], composite[w, SUCC]]],
    equal[composite[w, id[omega]], iterate[u, v]],
    {u -> composite[SUCC, SUCC], v -> singleton[0],
        w -> composite[NATADD, DUP]}]
```

```
equal[composite[NATADD, DUP], iterate[composite[SUCC, SUCC], singleton[0]]] == True
```

This justifies the rule:

```
iterate[composite[SUCC, SUCC], singleton[0]] := composite[NATADD, DUP]
```

On the multiplication side, we derive these lemmas:

```
Assoc[id[omega], NATMUL, LEFT[succ[singleton[0]]]]
```

```
composite[id[omega], iterate[composite[id[omega], SUCC, SUCC], singleton[0]]] ==
 iterate[composite[id[omega], SUCC, SUCC], singleton[0]]
```

```
composite[id[omega], iterate[composite[id[omega], SUCC, SUCC], singleton[0]]] :=
 iterate[composite[id[omega], SUCC, SUCC], singleton[0]]
```

```
Map[subclass[#, omega] &,
 ImageComp[id[omega], iterate[composite[id[omega], SUCC, SUCC], singleton[0]], V]]

subclass[range[iterate[composite[id[omega], SUCC, SUCC], singleton[0]]], omega] == True

subclass[range[iterate[composite[id[omega], SUCC, SUCC], singleton[0]]], omega] := True

SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
    {u -> range[iterate[composite[id[omega], SUCC, SUCC], singleton[0]]],
  v -> omega, w -> SUCC}]

subclass[image[SUCC, range[iterate[composite[id[omega], SUCC, SUCC], singleton[0]]]],
  omega] == True

subclass[image[SUCC,
   range[iterate[composite[id[omega], SUCC, SUCC], singleton[0]]]], omega] := True

SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
    {u -> image[SUCC, range[iterate[composite[id[omega], SUCC, SUCC], singleton[0]]]],
  v -> omega, w -> SUCC}]

subclass[image[SUCC, image[SUCC,
    range[iterate[composite[id[omega], SUCC, SUCC], singleton[0]]]]], omega] == True

subclass[image[SUCC, image[SUCC,
    range[iterate[composite[id[omega], SUCC, SUCC], singleton[0]]]]], omega] := True
```

The last step is to use uniqueness of **iterate** again:

```
SubstTest[implies, and[equal[image[w, singleton[0]], v],
        equal[composite[u, w], composite[w, SUCC]]],
    equal[composite[w, id[omega]], iterate[u, v]],
    {u -> composite[SUCC, SUCC], v -> singleton[0],
        w -> iterate[composite[id[omega], SUCC, SUCC], singleton[0]]}]

equal[composite[NATADD, DUP],
  iterate[composite[id[omega], SUCC, SUCC], singleton[0]]] == True

iterate[composite[id[omega], SUCC, SUCC], singleton[0]] := composite[NATADD, DUP]
```

This says that **2 x = x + x**:

```
composite[NATMUL, LEFT[succ[singleton[0]]]]

composite[NATADD, DUP]
```