left multiplication by x is one−to−one unless x = 0

Johan G. F. Belinfante
2002 October 5

```
<< goedel52.p74; << tools.m

:Package Title: goedel52.p74          2002 October 5 at 4:35 a.m.

It is now:  2002 Oct 5 at 22:25

Loading Simplification Rules

TOOLS.M                   Revised 2002 September 16

weightlimit = 40
```

■ summary

This notebook contains a derivation of the fact that left−multiplicationby nonzero numbers is one−to−one.The basic tool is the distributive law for multiplication over subtraction.

■ left multiplication

The following temporary abbreviation is convenient:

```
leftmul[x_] := composite[NATMUL, LEFT[x]]
```

The following fact about left−multiplicationwill be needed at some point:

```
composite[id[omega], inverse[LEFT[x]], inverse[NATMUL]] // DoubleInverse

composite[id[omega], inverse[LEFT[x]], inverse[NATMUL]] ==
 composite[inverse[LEFT[x]], inverse[NATMUL]]

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

■ distributive law for multiplication over subtraction

The distributibutive law for multiplication over subtraction was derived recently in the following form:

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

union[image[V, intersection[z, complement[y]]], natsub[natmul[x, y], natmul[x, z]]]
```

The first task will be to use **symdif** and **VSTriNormality** to rewrite this rule so that the variables **y** and **z** are eliminated.

```
symdif[composite[rotate[NATADD], cross[leftmul[x], leftmul[x]], id[inverse[S]]],
    composite[leftmul[x], rotate[NATADD]]] // VSTriNormality
```

```
union[composite[intersection[composite[complement[rotate[NATADD]],
     cross[composite[NATMUL, LEFT[x]], composite[NATMUL, LEFT[x]]]],
    composite[NATMUL, LEFT[x], rotate[NATADD]]], id[inverse[S]]],
  composite[intersection[composite[rotate[NATADD],
     cross[composite[NATMUL, LEFT[x]], composite[NATMUL, LEFT[x]]]],
    composite[complement[NATMUL], LEFT[x], rotate[NATADD]]], id[inverse[S]]]] == 0
```

```
union[composite[intersection[composite[complement[rotate[NATADD]],
     cross[composite[NATMUL, LEFT[x_]], composite[NATMUL, LEFT[x_]]]],
    composite[NATMUL, LEFT[x_], rotate[NATADD]]], id[inverse[S]]],
  composite[intersection[composite[rotate[NATADD],
     cross[composite[NATMUL, LEFT[x_]], composite[NATMUL, LEFT[x_]]]],
    composite[complement[NATMUL], LEFT[x_], rotate[NATADD]]], id[inverse[S]]]] := 0
```

```
SubstTest[equal, 0, symdif[u, v],
 {u -> composite[rotate[NATADD], cross[leftmul[x], leftmul[x]], id[inverse[S]]],
v ->     composite[leftmul[x], rotate[NATADD]]}]
```

```
True == equal[composite[NATMUL, LEFT[x], rotate[NATADD]], composite[rotate[NATADD],
   cross[composite[NATMUL, LEFT[x]], composite[NATMUL, LEFT[x]]], id[inverse[S]]]]
```

This reformulation of the distributive law will be made into a permanent rewrite rule:

```
composite[NATMUL, LEFT[x_], rotate[NATADD]] := composite[rotate[NATADD],
  cross[composite[NATMUL, LEFT[x]], composite[NATMUL, LEFT[x]]], id[inverse[S]]]
```

■ corollary

Lemma:

```
composite[FIRST, id[cart[V, union[intersection[omega, complement[image[V, x]]],
     intersection[image[V, intersection[omega, singleton[x]]], singleton[0]]]]],
  inverse[NATADD]] // VSNormality
```

```
composite[FIRST,
  id[cart[V, union[intersection[omega, complement[image[V, x]]], intersection[
      image[V, intersection[omega, singleton[x]]], singleton[0]]]]], inverse[NATADD]] ==
 union[composite[inverse[E], IMAGE[id[complement[image[V, x]]]], id[omega]],
  id[union[intersection[omega, complement[image[V, x]]],
    intersection[omega, image[V, intersection[omega, singleton[x]]]]]]]
```

```
composite[FIRST, id[cart[V, union[intersection[omega, complement[image[V, x_]]],
     intersection[image[V, intersection[omega, singleton[x_]]], singleton[0]]]]],
  inverse[NATADD]] := union[composite[inverse[E], IMAGE[id[complement[image[V, x]]]],
   id[omega]], id[union[intersection[omega, complement[image[V, x]]],
    intersection[omega, image[V, intersection[omega, singleton[x]]]]]]]
```

From the distrbutive law one obtains the following corollary by using **IminComp**:

```
Map[composite[inverse[#], id[image[V, x]]] &,
 IminComp[leftmul[x], rotate[NATADD], singleton[0]]]
```

```
composite[intersection[S,
   composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]]], id[image[V, x]]] ==
 id[intersection[omega, image[V, x], image[V, intersection[omega, singleton[x]]]]]
```

```
composite[intersection[S,
   composite[inverse[LEFT[x_]], inverse[NATMUL], NATMUL, LEFT[x_]]], id[image[V, x_]]] :=
 id[intersection[omega, image[V, x], image[V, intersection[omega, singleton[x]]]]]
```

The inverse will also be needed:

```
composite[id[image[V, x]],
  intersection[composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]],
   inverse[S]]] // DoubleInverse
```

```
composite[id[image[V, x]], intersection[
   composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]], inverse[S]]] ==
 id[intersection[omega, image[V, x], image[V, intersection[omega, singleton[x]]]]]
```

```
composite[id[image[V, x_]], intersection[
   composite[inverse[LEFT[x_]], inverse[NATMUL], NATMUL, LEFT[x_]], inverse[S]]] :=
 id[intersection[omega, image[V, x], image[V, intersection[omega, singleton[x]]]]]
```

■ eliminating S

The basic idea is to use dichotomy to eliminate **S** from the above formulas:

```
union[composite[id[omega], S, id[omega]], composite[id[omega], inverse[S], id[omega]]]
```

```
cart[omega, omega]
```

Lemma

```
AssInt[S, composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]],
  cart[omega, omega]] // Reverse
```

```
composite[id[omega], intersection[S,
   composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]]], id[omega]] ==
 intersection[S, composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]]]
```

```
composite[id[omega], intersection[S,
   composite[inverse[LEFT[x_]], inverse[NATMUL], NATMUL, LEFT[x_]]], id[omega]] :=
 intersection[S, composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]]]
```

The inverse will also be needed:

```
AssInt[inverse[S], composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]],
  cart[omega, omega]] // Reverse
```

```
composite[id[omega],
  intersection[composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]],
   inverse[S]], id[omega]] ==
 intersection[composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]], inverse[S]]
```

```
composite[id[omega], intersection[composite[inverse[LEFT[x_]],
    inverse[NATMUL], NATMUL, LEFT[x_]], inverse[S]], id[omega]] :=
 intersection[composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]], inverse[S]]
```

The dichotomy is used as follows:

```
SubstTest[intersection, w, union[u, v],
    {u -> composite[id[omega], S, id[omega]],
        v -> composite[id[omega], inverse[S], id[omega]],
        w -> composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]]}]

composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]] ==
 union[intersection[S, composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]]],
  intersection[
   composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]], inverse[S]]]
```

This can be cleaned up:

```
Map[composite[id[image[V, x]], #] &, %]

composite[id[image[V, x]], inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]] ==
 union[composite[id[image[V, x]],
   intersection[S, composite[inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]]]],
  id[intersection[omega, image[V, x], image[V, intersection[omega, singleton[x]]]]]]
```

The same trick is used a second time:

```
Map[composite[id[image[V, x]], inverse[#]] &, %]

composite[id[image[V, x]], inverse[LEFT[x]], inverse[NATMUL], NATMUL, LEFT[x]] ==
 id[intersection[omega, image[V, x], image[V, intersection[omega, singleton[x]]]]]

composite[id[image[V, x_]], inverse[LEFT[x_]], inverse[NATMUL], NATMUL, LEFT[x_]] :=
 id[intersection[omega, image[V, x], image[V, intersection[omega, singleton[x]]]]]
```

■ FUNCTION rule

A general rule:

```
FUNCTION[composite[id[image[V, x]], y]] // AssertTest

FUNCTION[composite[id[image[V, x]], y]] == or[equal[0, x], FUNCTION[composite[Id, y]]]

FUNCTION[composite[id[image[V, x_]], y_]] := or[equal[0, x], FUNCTION[composite[Id, y]]]
```

The final step is this:

```
SubstTest[subclass, composite[w, inverse[w]], Id,
  w -> composite[id[image[V, x]], inverse[LEFT[x]], inverse[NATMUL]]] // Reverse

or[equal[0, x], FUNCTION[composite[inverse[LEFT[x]], inverse[NATMUL]]]] == True
```

This says that left−multiplicationby nonzero numbers is one−to−one:

```
or[equal[0, x_], FUNCTION[composite[inverse[LEFT[x_]], inverse[NATMUL]]]] := True
```