# the integer zero

*Johan G. F. Belinfante*
*2002 December 1*

```
<< goedel52.q58; << tools.m
```

```
:Package Title: goedel52.q58          2002 November 30 at 10:57 p.m

It is now:  2002 Dec 1 at 15:24

Loading Simplification Rules

TOOLS.M                      Revised 2002 November 30

weightlimit = 40
```

## ■ summary

The integer zero is the identity function **id[omega]** on the set **omega** of all natural numbers. In general, integers are equivalence classes of ordered pairs of natural numbers with respect to the equivalence relation **EQUIDIFF**. If an ordered pair **pair[x,y]** belongs to the identity, the natural numbers **x** and **y** have zero difference. In earlier work it was shown that zero is an integer, that is, **id[omega]** belongs to the set **Z** of integers:

```
member[id[omega], Z]
```

```
True
```

In this notebook, this statement will be strengthened by showing that **id[x]** is not an integer unless **x** is **omega**.

## ■ the key idea

The basic strategy is to use inverse images of functions, in this case the function **IMAGE[DUP]**. This is the function that takes each set **x** to the identity relation on **x**.

```
lambda[x, id[x]]
```

```
IMAGE[DUP]
```

The connection with inverse images is this:

```
class[x, member[id[x], y]]
```

```
image[inverse[IMAGE[DUP]], y]
```

Although our eventual goal is to replace **y** with the set **Z** of all integers, it is easier to begin with the set **range[PLUS]** of positive (actually we should say non−negative)integers. This is the range of the function **PLUS** that takes each natural number **x** to the integer **plus[x]**.

```
member[pair[x, plus[x]], PLUS]
```

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

To speed things up, the **simplify** flag is turned off, which affects only a few conditional rewrite rules that are not needed here.

```
simplify = False;
```

The **RelnNormality** test is applied to a function whose range is the set of interest:

```
Map[range, composite[inverse[IMAGE[DUP]], PLUS] // RelnNormality]
```

```
image[inverse[IMAGE[DUP]], range[PLUS]] ==
 intersection[complement[image[V, intersection[omega,
     image[fix[composite[power[SUCC], Di, SECOND]], singleton[0]]]]], singleton[omega]]
```

This complicated result is not needed except to deduce an elementary corollary:

```
Map[subclass[#, singleton[omega]] &, %]
```

```
subclass[image[inverse[IMAGE[DUP]], range[PLUS]], singleton[omega]] == True
```

```
subclass[image[inverse[IMAGE[DUP]], range[PLUS]], singleton[omega]] := True
```

This is easier to understand when variables are introduced:

```
SubstTest[implies, and[member[x, y], subclass[y, z]], member[x, z],
    {y -> image[inverse[IMAGE[DUP]], range[PLUS]], z -> singleton[omega]}]
```

```
or[equal[omega, x], not[member[x, V]], not[member[id[x], range[PLUS]]]] == True
```

This is made into a temporary rewrite rule which will be cleaned up later.

```
or[equal[omega, x_], not[member[x_, V]], not[member[id[x_], range[PLUS]]]] := True
```

## ■ cleanup activities

The first step is to remove the redundant second literal that says **x** is a set. This is not needed since it is implied by the third literal.

```
SubstTest[implies, and[member[u, v], subclass[v, w]], member[u, w],
    {u -> id[x], v -> y, w -> V}]
```

```
or[member[x, V], not[member[id[x], y]]] == True
```

```
or[member[x_, V], not[member[id[x_], y_]]] := True
```

A bit of standard reasoning is used to remove the unwanted literal.

```
Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3], not[implies[p1, p3]],
        {p1 -> member[id[x], range[PLUS]], p2 -> member[x, V], p3 -> equal[omega, x]}]]
```

```
or[equal[omega, x], not[member[id[x], range[PLUS]]]] == True
```

```
or[equal[omega, x_], not[member[id[x_], range[PLUS]]]] := True
```

The reverse implication also holds:

```
SubstTest[implies, and[equal[u, v], member[v, w]], member[u, w],
    {u -> id[x], v -> id[omega], w -> range[PLUS]}]
```

```
or[member[id[x], range[PLUS]], not[equal[omega, x]]] == True
```

```
or[member[id[x_], range[PLUS]], not[equal[omega, x_]]] := True
```

Since the implication goes both ways, we obtain a statement of logical equivalence:

```
equiv[equal[omega, x], member[id[x], range[PLUS]]]
```

```
True
```

This is made into a permanent rewrite rule, which replaces a weaker existing rule.

```
member[id[x_], range[PLUS]] := equal[omega, x]
```

## ■ similar result for the set of negative integers

The result for negative integers follows from the result for positive ones, but again there is an unneeded literal.

```
member[id[x], image[INVERSE, range[PLUS]]]
```

```
and[equal[omega, x], member[x, V]]
```

The redundant literal can be disposed of using **AssertTest**.

```
or[member[x, V], not[equal[omega, x]]] // AssertTest
```

```
or[member[x, V], not[equal[omega, x]]] == True
```

```
or[member[x_, V], not[equal[omega, x_]]] := True
```

The following logical equivalence is now recognized:

```
equiv[and[equal[omega, x], member[x, V]], equal[omega, x]]
```

```
True
```

The expedient thing to do is to make this a temporary rewrite rule:

```
and[equal[omega, x_], member[x_, V]] := equal[omega, x]
```

The union of the set of non−negativeintegers and the set of non−positiveintegers is the set of all integers. This yields the final formula:

```
SubstTest[member, id[x], union[u, v],
    {u -> range[PLUS], v -> image[INVERSE, range[PLUS]]}]
```

```
member[id[x], Z] == equal[omega, x]
```

```
member[id[x_], Z] := equal[omega, x]
```

## ■ reformulation without variables

It is useful to restate the result obtained without variables. The object is to strengthen the following inclusion to an equation.

```
subclass[singleton[id[omega]], intersection[Z, P[Id]]]
```

```
True
```

The following logical equivalence is recognized to be valid, but there is no corresponding rewrite rule.

```
equiv[equal[x, id[fix[x]]], subclass[x, Id]]
```

```
True
```

This result is made into a temporary rewrite rule.

```
equal[x_, id[fix[x_]]] := subclass[x, Id]
```

Now a bit of reasoning is required:

```
SubstTest[implies, and[equal[x, y], member[x, z]], member[y, z],
    {y -> id[fix[x]], z -> Z}]
```

```
or[equal[omega, fix[x]], not[member[x, Z]], not[subclass[x, Id]]] == True
```

```
or[equal[omega, fix[x_]], not[member[x_, Z]], not[subclass[x_, Id]]] := True
```

```
SubstTest[implies, and[equal[x, y], equal[y, z]], equal[x, z],
    {y -> id[fix[x]], z -> id[omega]}]
```

```
or[equal[x, id[omega]], not[equal[omega, fix[x]]], not[subclass[x, Id]]] == True
```

```
or[equal[x_, id[omega]], not[equal[omega, fix[x_]]], not[subclass[x_, Id]]] := True
```

```
Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[and[p2, p3], p4],
        not[implies[and[p1, p2], p4]], {p1 -> member[x, Z],
   p2 -> subclass[x, Id], p3 -> equal[omega, fix[x]], p4 -> equal[x, id[omega]]}]]
```

```
or[equal[x, id[omega]], not[member[x, Z]], not[subclass[x, Id]]] == True
```

```
or[equal[x_, id[omega]], not[member[x_, Z]], not[subclass[x_, Id]]] := True
```

The variables are removed as follows:

```
Map[equal[V, #] &,
 union[complement[Z], complement[P[Id]], singleton[id[omega]]] // Normality]
```

```
subclass[intersection[Z, P[Id]], singleton[id[omega]]] == True
```

```
subclass[intersection[Z, P[Id]], singleton[id[omega]]] := True
```

The inclusion goes both ways, so we get an equation:

```
SubstTest[and, subclass[u, v], subclass[v, u],
    {u -> intersection[Z, P[Id]], v -> singleton[id[omega]]}]
```

```
True == equal[intersection[Z, P[Id]], singleton[id[omega]]]
```

This is the variable−freeformulation of the theorem:

```
intersection[Z, P[Id]] := singleton[id[omega]]
```

## ■ a corollary

A similar result holds for the set of positive numbers:

```
equal[intersection[Z, range[PLUS]], range[PLUS]]
```

```
True
```

```
intersection[Z, range[PLUS]] := range[PLUS]
```

```
AssInt[P[Id], Z, range[PLUS]]
```

```
intersection[P[Id], range[PLUS]] == singleton[id[omega]]
```