the integer zero

Johan G. F. Belinfante  
2002 December 1

\[
\text{summar}y
\]

The integer zero is the identity function \( \text{id}[\omega] \) 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 \( \text{EQUIDIFF} \). If an ordered pair \( \text{pair}[x, y] \) 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, \( \text{id}[\omega] \) belongs to the set \( \mathbb{Z} \) of integers:

\[
\text{member}[\text{id}[\omega], \mathbb{Z}]
\]

True

In this notebook, this statement will be strengthened by showing that \( \text{id}[x] \) is not an integer unless \( x \) is \( \omega \).

\[
\text{the key idea}
\]

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

\[
\text{lambda}[x, \text{id}[x]]
\]

\( \text{IMAGE}[\text{DUP}] \)

The connection with inverse images is this:

\[
\text{class}[x, \text{member}[\text{id}[x], y]]
\]

\( \text{image}[\text{inverse}[\text{IMAGE}[\text{DUP}]], y] \)

Although our eventual goal is to replace \( y \) with the set \( \mathbb{Z} \) of all integers, it is easier to begin with the set \( \text{range}[\text{PLUS}] \) of positive (actually we should say non–negative)integers. This is the range of the function \( \text{PLUS} \) that takes each natural number \( x \) to the integer \( \text{plus}[x] \).
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
The reverse implication also holds:

\[
\text{SubstTest[}\text{implies, and}\{\text{equal}[u, v], \text{member}[v, w]\}, \text{member}[u, w], \\
\{u \rightarrow \text{id}[x], v \rightarrow \text{id}[	ext{omega}], w \rightarrow \text{range}[\text{PLUS}]\}] \\
\text{or}[\text{member}[\text{id}[x], \text{range}[\text{PLUS}]], \text{not}[\text{equal}[\text{omega}, x]]] == \text{True} \\
\text{or}[\text{member}[\text{id}[x_\_], \text{range}[\text{PLUS}]], \text{not}[\text{equal}[\text{omega}, x_\_]]] == \text{True}
\]

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

\[
\text{equiv}[\text{equal}[\text{omega}, x], \text{member}[\text{id}[x], \text{range}[\text{PLUS}]])
\]

True

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

\[
\text{member}[\text{id}[x_\_], \text{range}[\text{PLUS}]] := \text{equal}[\text{omega}, x]
\]

### 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.

\[
\text{member}[\text{id}[x], \text{image}[\text{INVERSE}, \text{range}[\text{PLUS}])] \\
\text{and}[\text{equal}[\text{omega}, x], \text{member}[x, V]]
\]

The redundant literal can be disposed of using \text{AssertTest}.

\[
\text{or}[\text{member}[x, V], \text{not}[\text{equal}[\text{omega}, x]]] // \text{AssertTest} \\
\text{or}[\text{member}[x, V], \text{not}[\text{equal}[\text{omega}, x]]] == \text{True} \\
\text{or}[\text{member}[x_, V], \text{not}[\text{equal}[\text{omega}, x_\_]]] == \text{True}
\]

The following logical equivalence is now recognized:

\[
\text{equiv}[\text{and}[\text{equal}[\text{omega}, x], \text{member}[x, V]], \text{equal}[\text{omega}, x]]
\]

True

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

\[
\text{and}[\text{equal}[\text{omega}, x_\_], \text{member}[x_\_, V]] := \text{equal}[\text{omega}, x]
\]

The union of the set of non-negative integers and the set of non-positive integers is the set of all integers. This yields the final formula:

\[
\text{SubstTest[member, id[x], union[u, v],} \\
\{u \rightarrow \text{range}[\text{PLUS}], v \rightarrow \text{image}[\text{INVERSE}, \text{range}[\text{PLUS}]]\}) \\
\text{member}[id[x], Z] == \text{equal}[\text{omega}, x]
\]
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–free formulation of the theorem:

\[\text{intersection}[Z, P[\text{Id}]] :\! = \text{singleton}[\text{id}[\omega]]\]

**a corollary**

A similar result holds for the set of positive numbers:

\[\text{equal}[\text{intersection}[Z, \text{range[PLUS]]], \text{range[PLUS]]}\]

True

\[\text{intersection}[Z, \text{range[PLUS]]} :\! = \text{range[PLUS]}\]

\[\text{assInt}[P[\text{Id}], Z, \text{range[PLUS]}]\]

\[\text{intersection}[P[\text{Id}], \text{range[PLUS]}] == \text{singleton}[\text{id}[\omega]]\]