range[PLUS]

Johan G. F. Belinfante
2006 September 5


```
In[1]:= SetDirectory["l:"]; << goedel85.04a; << tools.m

        :Package Title: goedel85.04a              2006 September 4 at 6:20 p.m.

        It is now:  2006 Sep 5 at 13:48

        Loading Simplification Rules

        TOOLS.M                        Revised 2006 August 22

        weightlimit = 40
```

---

summary

One strategy for proving theorems about integers is to consider separately the cases of positive and negative integers. In the **GOEDEL** program, a typical positive integer is **plus[nat[x]]**, and a typical negative integer is **inverse[plus[nat[y]]]**. Consequently, the strategy of proving theorems by considering separately positive and negative integers boils down to proving facts about the function **plus[nat[x]]**. This technique is somewhat akin to the use of wrappers, such as the wrapper **nat** for natural numbers. If one thinks of **plus** as a wrapper, it is natural to ask how to remove this wrapper. Many theorems about positive integers can be viewed as a statement that **plus[x]** belongs to some specified class **y** either for all **x**, or at least when **x** is a natural number. One way to remove the wrapper **plus** from such statements is to use a normality test for **image[inverse[PLUS], y]**. Another method is to make use of **reify**. A final possibility is to try imitating the most common method for removing the **nat** wrapper as well as many others. In the case of the **nat** wrapper, one can replace the wrapper with a numberhood literal by using the rewrite rule

```
In[2]:= equal[x, nat[x]]

Out[2]= member[x, omega]
```

In this notebook, a somewhat analogous rewrite rule is derived for the statement **equal[x, plus[APPLY[x, 0]]**. This technique is illustrated with an example in which integers are considered as subsets of the natural number plane **cart[omega, omega]**. In particular, the integer zero is the subset **id[omega]**. The binary operation of vector addition applied to subsets of this plane resembles but is technically different from integer addition. It is shown that any integer is equal to the vector sum of itself and the integer zero. In addition, this notebook illustrates the derivation of theorems by providing only partial proofs, relying on the rewrite rules of the **GOEDEL** program to fill in the missing steps. In many cases, by deliberately omitting steps of the proof, the execution time for the derivation is drastically reduced.

using plus[x] to prove theorems about integers

Lemma.

```
In[3]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]], {p1 → member[x, Z],
          p2 -> subclass[x, cart[omega, omega]], p3 -> subclass[x, cart[V, V]]}]]

Out[3]= or[not[member[x, Z]], subclass[x, cart[V, V]]] == True

In[4]:= or[not[member[x_, Z]], subclass[x_, cart[V, V]]] := True
```

The following theorem can be used to reduce statements about integers to statements about the set **range[PLUS]**.

```
In[5]:= Map[implies[member[x, Z], #] &, SubstTest[member, x, union[u, v],
          {u → range[PLUS], v → image[INVERSE, range[PLUS]]}]] // MapNotNot // Reverse

Out[5]= or[member[x, range[PLUS]], member[inverse[x], range[PLUS]], not[member[x, Z]]] == True

In[6]:= or[member[x_, range[PLUS]], member[inverse[x_], range[PLUS]], not[member[x_, Z]]] := True
```

domains of positive integers

Lemma.

```
In[10]:= Map[implies[member[x, range[PLUS]], member[x, #]] &,
          ImageComp[PLUS, inverse[PLUS], image[inverse[IMAGE[FIRST]], set[omega]]]]

Out[10]= or[equal[omega, domain[x]], not[member[x, range[PLUS]]]] == True

In[11]:= or[equal[omega, domain[x_]], not[member[x_, range[PLUS]]]] := True
```

Lemma

```
In[13]:= Map[not,
          SubstTest[and, implies[p1, p2], not[implies[p1, p3]], {p1 -> member[x, range[PLUS]],
            p2 -> equal[omega, domain[x]], p3 → member[0, domain[x]]}]]

Out[13]= or[member[0, domain[x]], not[member[x, range[PLUS]]]] == True

In[14]:= or[member[0, domain[x_]], not[member[x_, range[PLUS]]]] := True
```

APPLY theorem

Lemma.

```
In[21]:= Map[implies[member[x, range[PLUS]], member[x, #]] &,
          ImageComp[PLUS, inverse[PLUS], P[complement[cart[set[0], complement[omega]]]]]]

Out[21]= or[not[member[x, range[PLUS]]], subclass[image[x, set[0]], omega]] == True

In[22]:= or[not[member[x_, range[PLUS]]], subclass[image[x_, set[0]], omega]] := True
```

Lemma.

```
In[23]:= image[inverse[PLUS], FUNS] // Normality

Out[23]= image[inverse[PLUS], FUNS] == omega

In[24]:= image[inverse[PLUS], FUNS] := omega
```

Corollary.

```
In[25]:= Map[subclass[range[PLUS], #] &, ImageComp[PLUS, inverse[PLUS], FUNS]]

Out[25]= subclass[range[PLUS], FUNS] == True

In[26]:= subclass[range[PLUS], FUNS] := True
```

Corollary.

```
In[27]:= SubstTest[implies, and[member[x, y], subclass[y, z]],
          member[x, z], {y → range[PLUS], z → FUNS}]

Out[27]= or[FUNCTION[x], not[member[x, range[PLUS]]]] == True

In[28]:= or[FUNCTION[x_], not[member[x_, range[PLUS]]]] := True
```

Theorem.

```
In[29]:= Map[not, SubstTest[and, implies[p1, p2],
            implies[p1, p3], implies[p1, p4], not[implies[p1, p5]],
            {p1 -> member[x, range[PLUS]], p2 -> subclass[image[x, set[0]], omega],
             p3 → FUNCTION[x], p4 → member[0, domain[x]], p5 -> member[APPLY[x, 0], omega]}]]

Out[29]= or[member[APPLY[x, 0], omega], not[member[x, range[PLUS]]]] == True

In[30]:= or[member[APPLY[x_, 0], omega], not[member[x_, range[PLUS]]]] := True

In[55]:= SubstTest[implies, member[s, t], not[empty[t]],
          {s -> PAIR[0, APPLY[x, 0]], t -> intersection[x, plus[APPLY[x, 0]]]}]

Out[55]= or[not[equal[0, intersection[x, plus[APPLY[x, 0]]]]], not[member[0, domain[x]]],
            not[member[APPLY[x, 0], omega]], not[member[pair[0, APPLY[x, 0]], x]]] == True

In[56]:= (% /. x → x_) /. Equal → SetDelayed
```

```
In[61]:= SubstTest[implies, and[member[x, Z], member[y, Z]],
          or[disjoint[x, y], equal[x, y]], y → plus[APPLY[x, 0]]]

Out[61]= or[equal[0, intersection[x, plus[APPLY[x, 0]]]], equal[x, plus[APPLY[x, 0]]],
           not[member[x, Z]], not[member[APPLY[x, 0], omega]]] == True
```

```
In[62]:= (% /. x → x_) /. Equal → SetDelayed
```

The execution time for the following derivation was reduced from 21 seconds to 3 seconds by deliberately leaving out several steps of the proof. The omitted steps are:
implies[p2,p3], **implies[p1,p5]**, **implies[p1,p7]**, and **implies[and[p2,p4,p6],p8]**.

```
In[88]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p4], implies[and[p4, p5], p6],
            implies[and[p3, p7, p8], p9], not[implies[p1, p9]], {p1 → member[x, range[PLUS]],
             p2 -> member[APPLY[x, 0], omega], p3 → member[plus[APPLY[x, 0]], Z],
             p4 → member[0, domain[x]], p5 → FUNCTION[x], p6 → member[PAIR[0, APPLY[x, 0]], x],
             p7 → member[x, Z], p8 → not[disjoint[x, plus[APPLY[x, 0]]]],
             p9 → equal[x, plus[APPLY[x, 0]]]}]]

Out[88]= or[equal[x, plus[APPLY[x, 0]]], not[member[x, range[PLUS]]]] == True
```

```
In[89]:= or[equal[x_, plus[APPLY[x_, 0]]], not[member[x_, range[PLUS]]]] := True
```

```
In[90]:= equiv[equal[x, plus[APPLY[x, 0]]], member[x, union[set[0], range[PLUS]]]] // not // not

Out[90]= True
```

```
In[91]:= equal[x_, plus[APPLY[x_, 0]]] := or[equal[0, x], member[x, range[PLUS]]]
```

The remainder of this notebook is about a somewhat specialized application that illustrates this technique.

an example: vector addition of subsets of cart[omega,omega]

The set **Z** of integers has been constructed in the **GOEDEL** program as the equivalence classes of a relation **EQUIDIFF**. These equivalence classes are the lines in the plane of natural numbers which are parallel to **id[omega]**. The line **id[omega]** itself represents the integer zero. The lines above **id[omega]** represent positive integers, and those below represent negative integers. Vector addition of subsets of the plane is commutative and associatve. The operation of vector addition of subsets of the plane **cart[omega, omega]** is not quite the same as integer addition. The problem is that the vector sum of two lines parallel to **id[omega]** in the number plane may be only a subset of such a line, rather than the complete line. In order to obtain a complete line, one needs to use the function **HULL[Z]**, which transforms any non-empty subset of an integer to the unique integer that contains it.

review: vector addition in the natural number plane

The points **PAIR[x,y]** of the plane **cart[omega,omega]** can be thought of as vectors. The associative binary operation of vector addition is the direct product of **NATADD** with itself.

```
In[92]:= member[composite[cross[NATADD, NATADD], TWIST], SEMIGPS]

Out[92]= True
```

By definition, the direct product is the composite of the cross product with the function **TWIST**. The twist is needed to interchange **x2** and **y1** so that the first component of the vector sum is the sum of the first components of the individual vectors, and similarly for the second components.

```
In[93]:= APPLY[composite[cross[NATADD, NATADD], TWIST], PAIR[PAIR[x1, x2], PAIR[y1, y2]]]

Out[93]= PAIR[natadd[x1, y1], natadd[x2, y2]]
```

vector addition of subsets of the natural number plane

For any binary operation on a set, there is a corresponding binary operation on subsets. The vector sum of subsets **x** and **y** of the number plane is the set of all vector sums **u + v** where **u** belongs to **x** and **v** belongs to **y**. This associative binary operation of vector addition of subsets of the natural number plane **cart[omega, omega]** is the restriction of the function **composite[IMAGE[cross[NATADD,NATADD]],CROSS]** to the power set of **cart[omega, omega]**.

```
In[94]:= member[composite[IMAGE[cross[NATADD, NATADD]], CROSS,
           id[cart[P[cart[omega, omega]], P[cart[omega, omega]]]]], SEMIGPS]

Out[94]= True
```

Explicitly, if **x** and **y** are subsets of the natural number plane **cart[omega, omega]**, then their vector sum is the subset **composite[NATADD, cross[x,y], inverse[NATADD]]**.

```
In[95]:= APPLY[composite[IMAGE[cross[NATADD, NATADD]], CROSS], PAIR[setpart[x], setpart[y]]]

Out[95]= composite[NATADD, cross[setpart[x], setpart[y]], inverse[NATADD]]
```

Vector addition of subsets is commutative:

```
In[96]:= Map[equal[composite[NATADD, cross[y, x], inverse[NATADD]], composite[NATADD, #]] &,
          Assoc[cross[x, y], SWAP, inverse[NATADD]]]

Out[96]= equal[composite[NATADD, cross[x, y], inverse[NATADD]],
           composite[NATADD, cross[y, x], inverse[NATADD]]] == True

In[97]:= equal[composite[NATADD, cross[x_, y_], inverse[NATADD]],
           composite[NATADD, cross[y_, x_], inverse[NATADD]]] := True
```

By eliminating the variables, one can formulate this commutative law as a rewrite rule.

```
In[98]:= SubstTest[implies, equal[x, composite[x, SWAP]],
          equal[composite[IMAGE[x], CART], composite[IMAGE[x], CART, SWAP]],
          x → composite[cross[NATADD, NATADD], TWIST]]

Out[98]= equal[composite[IMAGE[cross[NATADD, NATADD]], CROSS],
           composite[IMAGE[cross[NATADD, NATADD]], CROSS, SWAP]] == True
```

```
In[99]:= composite[IMAGE[cross[NATADD, NATADD]], CROSS, SWAP] :=
          composite[IMAGE[cross[NATADD, NATADD]], CROSS]
```

If **x** is a subset of **cart[omega, omega]**, then so is **inverse[x]**. The inverse of a vector sum is the vector sum of their inverses.

```
In[100]:=
        inverse[composite[NATADD, cross[x, y], inverse[NATADD]]]

Out[100]=
        composite[NATADD, cross[inverse[x], inverse[y]], inverse[NATADD]]
```

For the vector sum of two integers **x + y**, there are in principle four cases to consider, because each of the two integers could be either positive or negative. Since negative integers are constructed as inverses of positive integers, this symmetry of vector addition permits the reduction of these four cases to just two: the vector sum of two positive integers, and the vector sum of a positive and a negative integer. The integer zero is the subset **id[omega]** in the natural number plane. In this notebook it is shown that the vector sum of any integer **z** with **id[omega]** is equal to **z**. The proof considers separately the case of positive integers and negative integers. It should be noted that one cannot replace **z** here with an arbitrary subset of **cart[omega, omega]**. In addition, the set **id[omega]** will be replaced for convenience with the global identity **Id**. This does not matter because the points of **Id** outside the natural number plane do not contribute anything to the vector sum.

vector addition of a positive integer and the integer zero

For the sum of a positive integer and the integer zero, it is convenient to use the following property of the function **plus[x]**.

```
In[101]:=
        Assoc[plus[x], NATADD, inverse[NATADD]] // Reverse

Out[101]=
        composite[NATADD, cross[Id, plus[x]], inverse[NATADD]] == plus[x]

In[102]:=
        composite[NATADD, cross[Id, plus[x_]], inverse[NATADD]] := plus[x]
```

Thinking of **plus** as a wrapper, one can remove the wrapper using the **APPLY** theorem as follows:

```
In[103]:=
        SubstTest[implies, equal[x, plus[w]], equal[
           composite[NATADD, cross[Id, x], inverse[NATADD]], x], w → APPLY[x, 0]] // MapNotNot

Out[103]=
        or[equal[x, composite[NATADD, cross[Id, x], inverse[NATADD]]],
          not[member[x, range[PLUS]]]] == True

In[109]:=
        or[equal[x_, composite[NATADD, cross[Id, x_], inverse[NATADD]]],
          not[member[x_, range[PLUS]]]] := True
```

For the case of negative integers, one has:

In[110]:=
```
composite[NATADD, cross[Id, inverse[plus[x]]], inverse[NATADD]] // DoubleInverse
```

Out[110]=
```
composite[NATADD, cross[Id, inverse[plus[x]]], inverse[NATADD]] == inverse[plus[x]]
```

In[111]:=
```
composite[NATADD, cross[Id, inverse[plus[x_]]], inverse[NATADD]] := inverse[plus[x]]
```

Using the **APPLY** theorem yields:

In[112]:=
```
SubstTest[implies, equal[x, plus[w]],
  equal[composite[NATADD, cross[Id, inverse[x]], inverse[NATADD]], inverse[x]],
  w → APPLY[x, 0]] // MapNotNot
```

Out[112]=
```
or[equal[composite[NATADD, cross[Id, inverse[x]], inverse[NATADD]], inverse[x]],
  not[member[x, range[PLUS]]]] == True
```

In[113]:=
```
or[equal[composite[NATADD, cross[Id, inverse[x_]], inverse[NATADD]], inverse[x_]],
  not[member[x_, range[PLUS]]]] := True
```

Lemma.

In[114]:=
```
SubstTest[implies, member[y, range[PLUS]], equal[inverse[y],
  composite[NATADD, cross[Id, inverse[y]], inverse[NATADD]]], y → inverse[x]]
```

Out[114]=
```
or[equal[composite[Id, x], composite[NATADD, cross[Id, x], inverse[NATADD]]],
  not[member[inverse[x], range[PLUS]]]] == True
```

In[115]:=
```
(% /. x → x_) /. Equal → SetDelayed
```

The main theorem combines the cases of positive integers and negative integers.

In[116]:=
```
Map[not, SubstTest[and, implies[p1, or[p2, p3]], implies[p2, p6],
  implies[p1, p4], implies[p3, p5], implies[and[p4, p5], p6],
  not[implies[p1, p6]], {p1 → member[x, Z], p2 → member[x, range[PLUS]],
   p3 → member[inverse[x], range[PLUS]], p4 → subclass[x, cart[V, V]],
   p5 -> equal[composite[Id, x], composite[NATADD, cross[Id, x], inverse[NATADD]]],
   p6 -> equal[x, composite[NATADD, cross[Id, x], inverse[NATADD]]]}]]
```

Out[116]=
```
or[equal[x, composite[NATADD, cross[Id, x], inverse[NATADD]]], not[member[x, Z]]] == True
```

In[117]:=
```
or[equal[x_, composite[NATADD, cross[Id, x_], inverse[NATADD]]],
  not[member[x_, Z]]] := True
```

On account of the commutativity of vector addition, one also has:

In[121]:=

```
Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
  {p1 → member[x, Z], p2 -> equal[x, composite[NATADD, cross[Id, x], inverse[NATADD]]],
   p3 -> equal[x, composite[NATADD, cross[x, Id], inverse[NATADD]]]}]]
```

Out[121]=

```
or[equal[x, composite[NATADD, cross[x, Id], inverse[NATADD]]], not[member[x, Z]]] == True
```

In[125]:=

```
or[equal[x_, composite[NATADD, cross[x_, Id], inverse[NATADD]]],
  not[member[x_, Z]]] := True
```

Comment. Here again, one step of the proof has been deliberately omitted: **implies[p2, p3]**. In this case, however, the execution time has not been significantly diminished. Whether or not one includes this step, the execution time is about 0.016 seconds.