range[VERTSECT[DIV]]

Johan G. F. Belinfante
2005 August 20; restored 2005 September 8 after hard disk crash

```
In[1]:= SetDirectory["l:"]; << goedel72.18a; << tools.m

        :Package Title: goedel72.18a              2005 August 18 at 11:30 p.m.

        It is now:  2005 Sep 8 at 14:32

        Loading Simplification Rules

        TOOLS.M                       Revised 2005 August 16

        weightlimit = 40
```

summary

The ring of integers is a principal ideal domain. Although the set **omega** of natural numbers is not a ring, a similar result holds for **omega**. In this notebook it is shown that any nonempty set of natural numbers which is closed under both addition and subtraction is equal to the set of multiples of some number. To be sure, subtraction is not always possible for natural numbers. A set of numbers is considered to be closed under subtraction if for any two members **x** and **y**, if **x – y** makes sense, then **x – y** is also a member of the set.

range[VERTSECT[DIV]]

The set **range[VERTSECT[DIV]]** is a rough analog for **omega** of the set of principal ideals of the ring of integers. The set of multiples of any natural number belongs to this set, as does the empty set.

```
In[5]:= Map[implies[member[x, omega], #] &,
          SubstTest[subclass, image[u, v], range[u], {u → VERTSECT[DIV], v → set[x]}]]

Out[5]= or[member[image[DIV, set[x]], range[VERTSECT[DIV]]], not[member[x, omega]]] == True

In[6]:= (% /. x → x_) /. Equal → SetDelayed
```

The numberhood literal is not needed.

```
In[7]:= SubstTest[implies, equal[0, y], member[y, range[VERTSECT[DIV]]], y → image[DIV, set[x]]]

Out[7]= or[member[x, omega], member[image[DIV, set[x]], range[VERTSECT[DIV]]]] == True

In[8]:= (% /. x → x_) /. Equal → SetDelayed
```

Combining the two cases yields:

```
In[9]:= SubstTest[and, implies[p1, p2], or[p1, p2], {p1 -> member[x, omega],
          p2 -> member[image[DIV, set[x]], range[VERTSECT[DIV]]]}] // Reverse
```

```
Out[9]= member[image[DIV, set[x]], range[VERTSECT[DIV]]] == True
```

```
In[10]:= member[image[DIV, set[x_]], range[VERTSECT[DIV]]] := True
```

Two special cases of interest:

```
In[11]:= member[0, range[VERTSECT[DIV]]]
```

```
Out[11]= True
```

```
In[12]:= SubstTest[member, image[DIV, set[x]], range[VERTSECT[DIV]], x → 0]
```

```
Out[12]= member[set[0], range[VERTSECT[DIV]]] == True
```

```
In[13]:= member[set[0], range[VERTSECT[DIV]]] := True
```

inclusions in one direction

In this section, two easy inclusions are derived.

```
In[14]:= Map[subclass[range[VERTSECT[DIV]], image[VERTSECT[DIV], #]] &, SubstTest[class,
           x, member[image[u, set[x]], v], {u → DIV, v → binclosed[NATADD]}]] // Reverse
```

```
Out[14]= subclass[range[VERTSECT[DIV]], binclosed[NATADD]] == True
```

```
In[15]:= subclass[range[VERTSECT[DIV]], binclosed[NATADD]] := True
```

```
In[16]:= Map[subclass[range[VERTSECT[DIV]], image[VERTSECT[DIV], #]] &, SubstTest[class, x,
           member[image[u, set[x]], v], {u → DIV, v → binclosed[rotate[NATADD]]}]] // Reverse
```

```
Out[16]= subclass[range[VERTSECT[DIV]], binclosed[rotate[NATADD]]] == True
```

```
In[17]:= subclass[range[VERTSECT[DIV]], binclosed[rotate[NATADD]]] := True
```

presence of 0

Lemma.

```
In[18]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
           {u → set[PAIR[nat[x], nat[x]]], v → cart[y, y], w → rotate[NATADD]}]
```

```
Out[18]= or[member[0, image[image[inverse[NATADD], y], y]], not[member[nat[x], y]]] == True
```

```
In[19]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

If a subtraction-invariant class holds any number at all, it must hold **0**.

```
In[20]:= Map[not, SubstTest[and, implies[p1, p3],
           implies[and[p2, p3], p4], not[implies[and[p1, p2], p4]],
           {p1 → member[nat[x], y], p2 → subclass[image[image[inverse[NATADD], y], y], y],
            p3 -> member[0, image[image[inverse[NATADD], y], y]], p4 → member[0, y]}]]

Out[20]= or[member[0, y], not[member[nat[x], y]],
           not[subclass[image[image[inverse[NATADD], y], y], y]] == True

In[21]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Remove the **nat** wrapper.

```
In[22]:= SubstTest[implies, equal[x, nat[z]], or[member[0, y], not[member[x, y]],
           not[subclass[image[image[inverse[NATADD], y], y], y]]], z → x]

Out[22]= or[member[0, y], not[member[x, omega]], not[member[x, y]],
           not[subclass[image[image[inverse[NATADD], y], y], y]] == True

In[23]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Next eliminate the numerical variable.

```
In[24]:= Map[equal[V, #] &, SubstTest[class, x,
            or[member[0, y], not[member[x, y]], not[member[x, omega]], not[subclass[z, y]]],
            z → image[image[inverse[NATADD], y], y]]] // Reverse

Out[24]= or[equal[0, intersection[omega, y]], member[0, y],
           not[subclass[image[image[inverse[NATADD], y], y], y]] == True

In[25]:= (% /. y → y_) /. Equal → SetDelayed
```

Corollary. If a nonempty set of numbers is closed under subtraction, it holds the number **0**.

```
In[26]:= Map[not, SubstTest[and, implies[and[p1, p3], p4], implies[and[p2, p4], p5],
           not[implies[and[p1, p2, p3], p5]], {p1 -> subclass[x, omega],
            p2 -> subclass[image[image[inverse[NATADD], x], x], x], p3 → not[equal[0, x]],
            p4 → not[equal[0, intersection[omega, x]]], p5 -> member[0, x]}]]

Out[26]= or[equal[0, x], member[0, x], not[subclass[x, omega]],
           not[subclass[image[image[inverse[NATADD], x], x], x]] == True

In[27]:= or[equal[0, x_], member[0, x_], not[subclass[x_, omega]],
           not[subclass[image[image[inverse[NATADD], x_], x_], x_]]] := True
```

a consequence of closure under addition

Lemma.

```
In[28]:= Map[implies[member[x, y], #] &,
          SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
            {u → cart[y, set[x]], v → cart[y, y], w → z}] // MapNotNot]

Out[28]= or[not[member[x, y]], subclass[image[z, cart[y, set[x]]], image[z, cart[y, y]]]] == True

In[29]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

If **x** belongs to a class **y** that is closed under a binary operation **z**, then **y** is invariant under **composite[z, RIGHT[x]]**.

```
In[30]:= Map[not, SubstTest[and, implies[p1, p3],
           implies[and[p2, p3], p4], not[implies[and[p1, p2], p4]],
           {p1 → member[x, y], p2 → subclass[image[z, cart[y, y]], y],
            p3 -> subclass[image[z, cart[y, set[x]]], image[z, cart[y, y]]],
            p4 -> subclass[image[z, cart[y, set[x]]], y]}]]

Out[30]= or[not[member[x, y]], not[subclass[image[z, cart[y, y]], y]],
           subclass[image[z, cart[y, set[x]]], y]] == True

In[31]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

This holds in particular for the case **z = NATADD**.

```
In[32]:= SubstTest[or, not[member[x, y]], not[subclass[image[z, cart[y, y]], y]],
          subclass[image[z, cart[y, set[x]]], y], z → NATADD]

Out[32]= or[not[member[x, y]], not[subclass[image[NATADD, cart[y, y]], y]],
           subclass[image[plus[x], y], y]] == True

In[33]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

If a set **w** is invariant under **u** and contains **v**, then it contains **range[iterate[u,v]]**. An application of this yields:

```
In[34]:= SubstTest[implies, and[invariant[u, w], subclass[v, w]],
          subclass[range[iterate[u, v]], w], {u → plus[nat[x]], v → set[0]}]

Out[34]= or[not[member[0, w]], not[subclass[image[plus[nat[x]], w], w]],
           subclass[image[DIV, set[nat[x]]], w]] == True

In[35]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

One can combine these results to conclude that if **y** is closed under addition and holds **0**, then **y** holds every multiple of any number in **y**.

```
In[36]:= Map[not, SubstTest[and, implies[and[p1, p2], p4],
           implies[and[p3, p4], p5], not[implies[and[p1, p2, p3], p5]],
           {p1 → member[nat[x], y], p2 -> subclass[image[NATADD, cart[y, y]], y],
            p3 → member[0, y], p4 -> subclass[image[plus[nat[x]], y], y],
            p5 -> subclass[image[DIV, set[nat[x]]], y]}]]

Out[36]= or[not[member[0, y]], not[member[nat[x], y]],
           not[subclass[image[NATADD, cart[y, y]], y]],
           subclass[image[DIV, set[nat[x]]], y]] == True
```

```
In[37]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Remove the **nat** wrapper.

```
In[38]:= SubstTest[implies, equal[x, nat[z]], or[not[member[0, y]], not[member[x, y]],
           not[subclass[image[NATADD, cart[y, y]], y]], subclass[image[DIV, set[x]], y]], z → x]

Out[38]= or[not[member[0, y]], not[member[x, omega]], not[member[x, y]],
           not[subclass[image[NATADD, cart[y, y]], y]], subclass[image[DIV, set[x]], y]] == True

In[39]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

---

an iterate formula

The next goal is to show that if a class is closed under subtraction of natural numbers, then it is also closed under forming remainders. This is true because remainders can be obtained by repeated subtraction. The uniqueness of iteration implies the following formula, which can be made into a rewrite rule by wrapping numbers with **nat** to avoid numberhood literals.

```
In[40]:= SubstTest[implies,
          and[equal[composite[w, SUCC], composite[u, w]], equal[image[w, set[0]], v]],
          equal[composite[w, id[omega]], iterate[u, v]],
          {u → inverse[plus[nat[x]]], v → set[nat[y]],
           w -> composite[image[inverse[NATADD], set[nat[y]]], times[nat[x]]]}]

Out[40]= equal[composite[image[inverse[NATADD], set[nat[y]]], times[nat[x]]],
           iterate[inverse[plus[nat[x]]], set[nat[y]]]] == True

In[41]:= iterate[inverse[plus[nat[x_]]], set[nat[y_]]] :=
          composite[image[inverse[NATADD], set[nat[y]]], times[nat[x]]]
```

The least remainder **natmod[nat[x],nat[y]]** is one of these remainders:

```
In[42]:= SubstTest[implies, and[member[u, v], subclass[v, w]],
          member[u, w], {u -> natmod[nat[x], nat[y]],
           v -> image[image[inverse[NATADD], set[nat[x]]], image[DIV, set[nat[y]]]]}]

Out[42]= or[member[natmod[nat[x], nat[y]], w], not[subclass[
             image[image[inverse[NATADD], set[nat[x]]], image[DIV, set[nat[y]]], w]]] == True

In[43]:= (% /. {w → w_, x -> x_, y → y_}) /. Equal → SetDelayed
```

The range of **iterate[u,v]** is the least class that is invariant under **u** and contains **v**. In the present case, this yields:

```
In[44]:= SubstTest[implies, and[invariant[u, w], subclass[v, w]],
          subclass[range[iterate[u, v]], w], {u → inverse[plus[nat[x]]], v → set[nat[y]]}]

Out[44]= or[not[member[nat[y], w]], not[subclass[image[inverse[plus[nat[x]]], w], w]], subclass[
             image[image[inverse[NATADD], set[nat[y]]], image[DIV, set[nat[x]]], w]] == True

In[45]:= (% /. {w → w_, x -> x_, y → y_}) /. Equal → SetDelayed
```

Combining the above facts yields the following result: if a class **w** is invariant under subtracting a particular number **nat[y]**, then for any member **nat[x]** of **w**, the class **w** must hold the least remainder obtained when **nat[x]** is divided by **nat[y]**.

```
In[46]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[p3, p4], not[implies[and[p1, p2], p4]],
           {p1 -> member[nat[x], w], p2 → invariant[inverse[plus[nat[y]]], w], p3 ->
             subclass[image[image[inverse[NATADD], set[nat[x]]], image[DIV, set[nat[y]]]], w],
            p4 → member[natmod[nat[x], nat[y]], w]}]]

Out[46]= or[member[natmod[nat[x], nat[y]], w], not[member[nat[x], w]],
           not[subclass[image[inverse[plus[nat[y]]], w], w]]] == True

In[47]:= (% /. {w → w_, x -> x_, y → y_}) /. Equal → SetDelayed
```

Theorem.

```
In[48]:= SubstTest[implies, and[member[x, y], subclass[image[z, cart[y, y]], y]],
          invariant[composite[z, RIGHT[x]], y], z → rotate[NATADD]]

Out[48]= or[not[member[x, y]], not[subclass[image[image[inverse[NATADD], y], y], y]],
           subclass[image[inverse[plus[x]], y], y]] == True

In[49]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Corollary:

```
In[50]:= Map[not, SubstTest[and, implies[and[p2, p3], p4], implies[and[p1, p4], p5],
           not[implies[and[p1, p2, p3], p5]], {p1 → member[nat[x], z],
            p2 -> member[nat[y], z], p3 -> subclass[image[image[inverse[NATADD], z], z], z],
            p4 → invariant[inverse[plus[nat[y]]], z], p5 → member[natmod[nat[x], nat[y]], z]}]]

Out[50]= or[member[natmod[nat[x], nat[y]], z], not[member[nat[x], z]], not[member[nat[y], z]],
           not[subclass[image[image[inverse[NATADD], z], z], z]]] == True

In[51]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

closure under NATMOD

The main result in this section is obtained by eliminating the variables in the result derived in the last section. To facilitate eliminating these variables, first the **nat** wrappers are removed.

```
In[52]:= SubstTest[implies, and[equal[x, nat[u]], equal[y, nat[v]]],
          or[member[natmod[x, y], z], not[member[x, z]], not[member[y, z]],
           not[subclass[image[image[inverse[NATADD], z], z], z]]], {u → x, v → y}]

Out[52]= or[member[natmod[x, y], z], not[member[x, omega]],
           not[member[x, z]], not[member[y, omega]], not[member[y, z]],
           not[subclass[image[image[inverse[NATADD], z], z], z]]] == True
```

```
In[53]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma.

```
In[54]:= member[pair[x, y], image[inverse[NATMOD], z]] // AssertTest

Out[54]= member[pair[x, y], image[inverse[NATMOD], z]] == member[natmod[x, y], z]

In[55]:= member[pair[x_, y_], image[inverse[NATMOD], z_]] := member[natmod[x, y], z]
```

The numerical variables are now removed:

```
In[56]:= Map[equal[0, composite[Id, complement[#]]] &, SubstTest[class, pair[x, y],
           implies[and[subclass[image[image[u, z], z], z], member[x, omega], member[y, omega],
             member[x, z], member[y, z]], member[pair[x, y], image[inverse[funpart[v]], z]]],
           {u → inverse[NATADD], v → NATMOD}]] // InvertFix // Reverse

Out[56]= or[not[subclass[image[image[inverse[NATADD], z], z], z]],
           subclass[cart[intersection[omega, z], intersection[omega, z]],
            image[inverse[NATMOD], z]]] == True

In[57]:= (% /. z → z_) /. Equal → SetDelayed
```

Lemma.

```
In[58]:= ImageComp[NATMOD, id[cart[omega, omega]], cart[z, z]] // Reverse

Out[58]= image[NATMOD, cart[intersection[omega, z], intersection[omega, z]]] ==
          image[NATMOD, cart[z, z]]

In[59]:= image[NATMOD, cart[intersection[omega, z_], intersection[omega, z_]]] :=
          image[NATMOD, cart[z, z]]
```

Lemma.

```
In[60]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u -> cart[intersection[omega, x], intersection[omega, x]],
           v -> image[inverse[NATMOD], x], w → NATMOD}]

Out[60]= or[not[subclass[cart[intersection[omega, x], intersection[omega, x]],
             image[inverse[NATMOD], x]]], subclass[image[NATMOD, cart[x, x]], x]] == True

In[61]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. Closure under **rotate[NATADD]** implies closure under **NATMOD**.

```
In[62]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 → subclass[image[image[inverse[NATADD], x], x], x], p2 -> subclass[
              cart[intersection[omega, x], intersection[omega, x]], image[inverse[NATMOD], x]],
            p3 → subclass[image[NATMOD, cart[x, x]], x]}]]

Out[62]= or[not[subclass[image[image[inverse[NATADD], x], x], x]],
           subclass[image[NATMOD, cart[x, x]], x]] == True
```

```
In[63]:= (% /. x → x_) /. Equal → SetDelayed
```

The remaining variable can also be removed:

```
In[64]:= Map[equal[V, #] &, SubstTest[class, x, implies[member[x, u], member[x, v]],
           {u → binclosed[rotate[NATADD]], v → binclosed[NATMOD]}]] // Reverse

Out[64]= subclass[binclosed[rotate[NATADD]], binclosed[NATMOD]] == True

In[65]:= subclass[binclosed[rotate[NATADD]], binclosed[NATMOD]] := True
```

least nonzero element

The idea is now to show that if a set of numbers is closed under both addition and subtraction, and if it holds some number other than **0**, then it is the set of multiples of its least nonzero element. Some lemmas are needed. First, the fact that any nonempty set of numbers has a least member.

```
In[66]:= Map[or[#, member[A[intersection[x, complement[set[0]]]], x]] &, SubstTest[implies,
           subclass[w, omega], or[empty[w], member[A[w], w]], w -> dif[x, set[0]]]]

Out[66]= or[member[A[intersection[x, complement[set[0]]]], x],
           not[subclass[x, omega]], subclass[x, set[0]]] == True

In[67]:= (% /. x → x_) /. Equal → SetDelayed
```

The least member is not zero.

```
In[68]:= SubstTest[implies, subclass[w, omega],
           or[empty[w], member[A[w], w]], w -> dif[x, set[0]]] // MapNotNot

Out[68]= or[not[equal[0, A[intersection[x, complement[set[0]]]]]],
           not[subclass[x, omega]], subclass[x, set[0]]] == True

In[69]:= (% /. x → x_) /. Equal → SetDelayed
```

If **x** is closed under addition and subtraction, and holds more than just **0**, then it contains the multiples of the least nonzero member.

```
In[70]:= Map[not, SubstTest[and, implies[and[p1, p3, p4], p6],
           implies[and[p4, p5], p7], implies[and[p1, p7], p8], implies[and[p1, p8], p9],
           implies[and[p2, p6, p8, p9], p10], not[implies[and[p1, p2, p3, p4, p5], p10]],
           {p1 → subclass[x, omega], p2 → subclass[image[NATADD, cart[x, x]], x],
            p3 -> subclass[image[image[inverse[NATADD], x], x], x], p4 → not[equal[0, x]],
            p5 → not[equal[set[0], x]], p6 → member[0, x], p7 → not[subclass[x, set[0]]],
            p8 -> member[A[intersection[x, complement[set[0]]]], x],
            p9 -> member[A[intersection[x, complement[set[0]]]], omega],
            p10 -> subclass[image[DIV, set[A[intersection[x, complement[set[0]]]]]], x]}]]

Out[70]= or[equal[0, x], equal[x, set[0]],
           not[subclass[x, omega]], not[subclass[image[NATADD, cart[x, x]], x]],
           not[subclass[image[image[inverse[NATADD], x], x], x]],
           subclass[image[DIV, set[A[intersection[x, complement[set[0]]]]]], x]] == True

In[71]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[72]:= SubstTest[implies, member[u, v], subclass[A[v], u],
          {u -> natmod[y, A[intersection[x, complement[set[0]]]]],
           v -> intersection[x, complement[set[0]]]}]

Out[72]= or[equal[0, A[intersection[x, complement[set[0]]]]],
           equal[0, natmod[y, A[intersection[x, complement[set[0]]]]]],
           not[member[y, omega]], not[member[A[intersection[x, complement[set[0]]]], omega]],
           not[member[natmod[y, A[intersection[x, complement[set[0]]]], x]]] == True

In[73]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The next step involves the following ideas. To save time, some bundling will be done.

```
In[74]:= {implies[and[p4, p5], p7], implies[and[p1, p7], p8],
           implies[and[p1, p8], p9], implies[and[p1, p7], p10], implies[and[p1, p11], p12],
           implies[and[p3, p8, p9, p11, p12], p13], implies[and[p8, p9, p10, p12, p13], p14],
           implies[p14, p15] (* ,implies[and[p1,p3,p4,p5,p11],p15] *)} /.
          {p1 → subclass[x, omega], p3 -> subclass[image[image[inverse[NATADD], x], x], x],
           p4 → not[equal[0, x]], p5 → not[equal[set[0], x]], p7 → not[subclass[x, set[0]]],
           p8 -> member[A[intersection[x, complement[set[0]]]], x],
           p9 -> member[A[intersection[x, complement[set[0]]]], omega],
           p10 → not[equal[0, A[intersection[x, complement[set[0]]]]]], p11 → member[y, x],
           p12 → member[y, omega],
           p13 → member[natmod[y, A[intersection[x, complement[set[0]]]], x],
           p14 → equal[0, natmod[y, A[intersection[x, complement[set[0]]]]]],
           p15 → member[pair[A[intersection[x, complement[set[0]]]], y], DIV]}

Out[74]= {True, True, True, True, True, True, True, True}
```

Even with bundling, the deduction takes quite a while, mainly in checking the validity of the reasoning.

```
In[75]:= Map[not,
          SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3], implies[and[p1, p3], p4],
           implies[and[p1, p2], p5], implies[p1, p6], implies[and[p1, p3, p4, p6], p7],
           implies[and[p3, p4, p5, p6, p7], p8], implies[p8, p9], not[implies[p1, p9]],
           {p1 → and[member[y, x], subclass[x, omega], subclass[image[image[inverse[NATADD], x],
                x], x], not[equal[0, x]], not[equal[set[0], x]]], p2 → not[subclass[x, set[0]]],
            p3 -> member[A[intersection[x, complement[set[0]]]], x],
            p4 -> member[A[intersection[x, complement[set[0]]]], omega],
            p5 → not[equal[0, A[intersection[x, complement[set[0]]]]]],
            p6 → member[y, omega], p7 → member[natmod[y, A[intersection[x, complement[set[0]]]]],
              x], p8 → equal[0, natmod[y, A[intersection[x, complement[set[0]]]]]],
            p9 → member[pair[A[intersection[x, complement[set[0]]]], y], DIV]}]]

Out[75]= or[equal[0, x], equal[x, set[0]],
           member[pair[A[intersection[x, complement[set[0]]]], y], DIV], not[member[y, x]],
           not[subclass[x, omega]], not[subclass[image[image[inverse[NATADD], x], x], x]]] == True

In[76]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The next step is to eliminate the variable **y**.

```
In[77]:= Map[equal[V, #] &,
           SubstTest[class, y, or[equal[0, x], equal[x, set[0]], member[pair[u, y], v],
             not[member[y, x]], not[subclass[x, omega]], not[subclass[w, x]]],
            {u -> A[intersection[x, complement[set[0]]]], v → DIV,
             w -> image[image[inverse[NATADD], x], x]}]] // Reverse

Out[77]= or[equal[x, set[0]], not[subclass[x, omega]],
           not[subclass[image[image[inverse[NATADD], x], x], x]],
           subclass[x, image[DIV, set[A[intersection[x, complement[set[0]]]]]]]] == True

In[78]:= (% /. x → x_) /. Equal → SetDelayed
```

Now put it all together:

```
In[79]:= SubstTest[and, implies[p1, subclass[x, y]],
            implies[p2, subclass[y, x]], {p1 → and[not[equal[x, set[0]]],
               subclass[x, omega], subclass[image[image[inverse[NATADD], x], x], x]],
             p2 → and[not[equal[0, x]], not[equal[x, set[0]]], subclass[x, omega],
               subclass[image[NATADD, cart[x, x]], x],
               subclass[image[image[inverse[NATADD], x], x], x]], y ->
              image[DIV, set[A[intersection[x, complement[set[0]]]]]]}] // MapNotNot // Reverse

Out[79]= or[equal[0, x], equal[x, image[DIV, set[A[intersection[x, complement[set[0]]]]]]],
           equal[x, set[0]], not[subclass[x, omega]],
           not[subclass[image[NATADD, cart[x, x]], x]],
           not[subclass[image[image[inverse[NATADD], x], x], x]]] == True

In[80]:= (% /. x → x_) /. Equal → SetDelayed
```

Three cases need to be combined: the case **x** is empty, the case **x = set[0]**, and the case that **x** has a least nonzero element.

```
In[81]:= Map[not, SubstTest[and, implies[p1, or[p2, p3, p4]], implies[p2, p5], implies[p3, p5],
           implies[p5, or[p2, p3]], implies[p4, or[p5, p6]], not[implies[p1, p6]],
           {p1 → and[member[x, V], subclass[x, omega], subclass[image[NATADD, cart[x, x]], x],
              subclass[image[image[inverse[NATADD], x], x], x]],
            p2 → equal[0, x], p3 -> equal[x, set[0]],
            p4 -> equal[x, image[DIV, set[A[intersection[x, complement[set[0]]]]]]],
            p5 → subclass[x, set[0]], p6 → member[x, range[VERTSECT[DIV]]]}]]
```

```
Out[81]= or[member[x, range[VERTSECT[DIV]]],
           not[subclass[x, omega]], not[subclass[image[NATADD, cart[x, x]], x]],
           not[subclass[image[image[inverse[NATADD], x], x], x]]] == True
```

```
In[82]:= (% /. x → x_) /. Equal → SetDelayed
```

The variable **x** is eliminated:

```
In[83]:= Map[equal[V, #] &, SubstTest[class, x, implies[member[x, u], member[x, v]],
            {u → intersection[P[omega], binclosed[NATADD], binclosed[rotate[NATADD]]],
             v → range[VERTSECT[DIV]]}]] // Reverse
```

```
Out[83]= subclass[intersection[binclosed[NATADD], binclosed[rotate[NATADD]], P[omega]],
           range[VERTSECT[DIV]]] == True
```

```
In[84]:= % /. Equal → SetDelayed
```

The reverse inclusion also holds, yielding an equation.

```
In[85]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u -> intersection[binclosed[NATADD], binclosed[rotate[NATADD]], P[omega]],
            v -> range[VERTSECT[DIV]]}]
```

```
Out[85]= True == equal[intersection[binclosed[NATADD], binclosed[rotate[NATADD]], P[omega]],
           range[VERTSECT[DIV]]]
```

This is the main result. Any set of numbers that is closed under both addition and subtraction is in the range of the function **VERTSECT[DIV]**.

```
In[86]:= intersection[binclosed[NATADD], binclosed[rotate[NATADD]], P[omega]] :=
           range[VERTSECT[DIV]]
```

a corollary

Since **binclosed[x]** is **Aclosed**, one finds:

```
In[87]:= SubstTest[Aclosure, binclosed[x],
           x → union[NATADD, rotate[NATADD], cart[id[complement[omega]], V]]]
```

```
Out[87]= Aclosure[range[VERTSECT[DIV]]] == range[VERTSECT[DIV]]
```

```
In[88]:= Aclosure[range[VERTSECT[DIV]]] := range[VERTSECT[DIV]]
```

In other words, **range[VERTSECT[DIV]]** is closed under arbitrary intersections.