an analog of induction for integers

Johan G. F. Belinfante
2006 July 27

```
In[1]:= SetDirectory["l:"]; << goedel83.23a; << tools.m

        :Package Title: goedel83.23a         2006 July 23 at 10:40 p.m.

        It is now:  2006 Jul 27 at 14:32

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 July 18

        weightlimit = 40
```

summary

The set **Z** of integers is constructed in the **GOEDEL** program as a set of functions whose graphs are contained in the cartesian square **cart[omega,omega]**, where **omega** is the set of natural numbers. The graphs of these functions form a set of parallel straight lines with slope one. One of these lines is the graph of **id[omega]** which is identified with the integer zero. The graphs of the functions representing positive integers are the lines that lie above zero, while the graphs for the negative integers are the lines below zero. If **x** is an integer, then so are **inverse[x]** and the function **composite[x, SUCC]**. An analog of induction for the set **Z** of integers is derived in this notebook: any set that holds **id[omega]** and is closed under taking inverses and under right composition with **SUCC** must contain the set **Z**.

closure under right composition with SUCC

Lemma.

```
In[2]:= SubstTest[member, plus[natadd[x, nat[y]]], Z, y → set[0]]

Out[2]= member[composite[SUCC, plus[x]], Z] == member[x, omega]

In[3]:= member[composite[SUCC, plus[x_]], Z] := member[x, omega]
```

Theorem.

```
In[4]:= Map[subclass[range[PLUS], image[PLUS, #]] &,
          SubstTest[class, x, implies[member[x, omega], member[x, z]], z -> image[
             inverse[PLUS], image[inverse[IMAGE[cross[inverse[SUCC], Id]]], Z]]]] // Reverse

Out[4]= subclass[image[IMAGE[cross[inverse[SUCC], Id]], range[PLUS]], Z] == True

In[5]:= % /. Equal → SetDelayed
```

Lemma.

```
In[6]:= SubstTest[member, composite[inverse[plus[x]], plus[nat[y]]], Z, y → set[0]]

Out[6]= member[composite[inverse[plus[x]], SUCC], Z] == member[x, omega]

In[7]:= member[composite[inverse[plus[x_]], SUCC], Z] := member[x, omega]
```

Theorem.

```
In[8]:= Map[subclass[range[PLUS], image[PLUS, #]] &,
          SubstTest[class, x, implies[member[x, omega], member[x, z]],
           z → image[inverse[PLUS], image[inverse[IMAGE[SWAP]],
              image[inverse[IMAGE[cross[inverse[SUCC], Id]]], Z]]]]] // Reverse

Out[8]= subclass[image[IMAGE[cross[inverse[SUCC], Id]], image[INVERSE, range[PLUS]]], Z] == True

In[9]:= % /. Equal → SetDelayed
```

Combining the two theorems, one obtains:

```
In[10]:= SubstTest[subclass, union[u, v], w, {u → range[PLUS], v -> image[INVERSE, range[PLUS]],
           w → image[inverse[IMAGE[cross[inverse[SUCC], Id]]], Z]}]

Out[10]= subclass[image[IMAGE[cross[inverse[SUCC], Id]], Z], Z] == True

In[11]:= subclass[image[IMAGE[cross[inverse[SUCC], Id]], Z], Z] := True
```

Corollary. The set **Z** is invariant under right composition with **SUCC**.

```
In[12]:= SubstTest[implies, and[member[x, y], subclass[y, z]], member[x, z],
           {y → Z, z → image[inverse[IMAGE[cross[inverse[SUCC], Id]]], Z]}]

Out[12]= or[member[composite[x, SUCC], Z], not[member[x, Z]]] == True

In[13]:= or[member[composite[x_, SUCC], Z], not[member[x_, Z]]] := True
```

an analog of induction for Z

The following lemma transfers the inductive property of **omega** to **range[PLUS]**.

```
In[14]:= SubstTest[implies, and[member[0, y], invariant[SUCC, y]],
           subclass[omega, y], y → image[inverse[PLUS], x]]

Out[14]= or[not[member[id[omega], x]],
           not[subclass[image[SUCC, image[inverse[PLUS], x]], image[inverse[PLUS], x]]],
           subclass[omega, image[inverse[PLUS], x]]] == True

In[15]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[16]:= symdif[composite[PLUS, SUCC],
            composite[IMAGE[cross[inverse[SUCC], Id]], PLUS]] // VSNormality

Out[16]= union[intersection[composite[PLUS, SUCC],
             composite[complement[IMAGE[cross[inverse[SUCC], Id]]], PLUS]],
           intersection[composite[complement[PLUS], SUCC],
            composite[IMAGE[cross[inverse[SUCC], Id]], PLUS]]] == 0

In[17]:= % /. Equal → SetDelayed
```

It is unclear how best to orient the following rewrite rule.

```
In[18]:= SubstTest[equal, 0, symdif[u, v],
           {u -> composite[PLUS, SUCC], v -> composite[IMAGE[cross[inverse[SUCC], Id]], PLUS]}]

Out[18]= True == equal[composite[PLUS, SUCC], composite[IMAGE[cross[inverse[SUCC], Id]], PLUS]]

In[19]:= composite[IMAGE[cross[inverse[SUCC], Id]], PLUS] := composite[PLUS, SUCC]
```

Corollary.

```
In[20]:= IminComp[IMAGE[cross[inverse[SUCC], Id]], PLUS, x] // Reverse

Out[20]= image[inverse[PLUS], image[inverse[IMAGE[cross[inverse[SUCC], Id]]], x]] ==
          image[inverse[SUCC], image[inverse[PLUS], x]]

In[21]:= image[inverse[PLUS], image[inverse[IMAGE[cross[inverse[SUCC], Id]]], x_]] :=
          image[inverse[SUCC], image[inverse[PLUS], x]]
```

Corollary.

```
In[22]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
           {u -> x, v -> image[inverse[IMAGE[cross[inverse[SUCC], Id]]], x], w → inverse[PLUS]}]

Out[22]= or[not[subclass[image[IMAGE[cross[inverse[SUCC], Id]], x], x]],
            subclass[image[SUCC, image[inverse[PLUS], x]], image[inverse[PLUS], x]]] == True

In[23]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[24]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
           {u → omega, v → image[inverse[PLUS], x], w → PLUS}]

Out[24]= or[not[subclass[omega, image[inverse[PLUS], x]]], subclass[range[PLUS], x]] == True

In[25]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[26]:= Map[not, SubstTest[and, implies[and[p1, p3], p4], implies[p2, p3],
           implies[p4, p5], not[implies[and[p1, p2], p5]], {p1 -> member[id[omega], x],
            p2 -> subclass[image[IMAGE[cross[inverse[SUCC], Id]], x], x],
            p3 -> subclass[image[SUCC, image[inverse[PLUS], x]], image[inverse[PLUS], x]],
            p4 -> subclass[omega, image[inverse[PLUS], x]], p5 → subclass[range[PLUS], x]}]]

Out[26]= or[not[member[id[omega], x]],
           not[subclass[image[IMAGE[cross[inverse[SUCC], Id]], x], x]],
           subclass[range[PLUS], x]] == True

In[27]:= (% /. x → x_) /. Equal → SetDelayed
```

The set of integers is the union of the set of positive integers and the set of negative integers.

```
In[28]:= Map[implies[#, subclass[Z, x]] &, SubstTest[subclass, union[u, v],
            x, {u → range[PLUS], v → image[INVERSE, range[PLUS]]}]] // Reverse

Out[28]= or[not[subclass[image[INVERSE, range[PLUS]], x]],
           not[subclass[range[PLUS], x]], subclass[Z, x]] == True

In[29]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[30]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u → range[PLUS], v → x, w → IMAGE[SWAP]}]

Out[30]= or[not[subclass[range[PLUS], x]],
           subclass[image[INVERSE, range[PLUS]], image[IMAGE[SWAP], x]]] == True

In[31]:= (% /. x → x_) /. Equal → SetDelayed

In[32]:= Map[not, SubstTest[and, implies[p2, p3], implies[and[p1, p3], p4],
           implies[and[p2, p4], p5], not[implies[and[p1, p2], p5]],
           {p1 -> invariant[IMAGE[SWAP], x], p2 -> subclass[range[PLUS], x],
            p3 -> subclass[image[INVERSE, range[PLUS]], image[IMAGE[SWAP], x]],
            p4 -> subclass[image[INVERSE, range[PLUS]], x], p5 → subclass[Z, x]}]]

Out[32]= or[not[subclass[image[IMAGE[SWAP], x], x]],
           not[subclass[range[PLUS], x]], subclass[Z, x]] == True

In[33]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. An analog of induction for the set **Z** of integers: if class x holds **id[omega]**, and is invariant under **IMAGE[-SWAP]** and **IMAGE[cross[inverse[SUCC],Id]]**, then **x** contains the set of integers.

```
In[34]:= Map[not, SubstTest[and, implies[and[p1, p2], p4],
           implies[and[p3, p4], p5], not[implies[and[p1, p2, p3], p5]],
           {p1 → member[id[omega], x], p2 → invariant[IMAGE[cross[inverse[SUCC], Id]], x],
            p3 → invariant[IMAGE[SWAP], x], p4 → subclass[range[PLUS], x], p5 → subclass[Z, x]}]]

Out[34]= or[not[member[id[omega], x]], not[subclass[image[IMAGE[SWAP], x], x]],
           not[subclass[image[IMAGE[cross[inverse[SUCC], Id]], x], x]], subclass[Z, x]] == True
```

```
In[35]:= or[not[member[id[omega], x_]], not[subclass[image[IMAGE[SWAP], x_], x_]], not[
           subclass[image[IMAGE[cross[inverse[SUCC], Id]], x_], x_]], subclass[Z, x_]] := True
```

variable-free formulation

Removing the variable **x** yields:

```
In[36]:= Map[equal[V, #] &,
           SubstTest[class, x, implies[and[member[u, x], member[x, v]], subclass[z, x]],
            {u → id[omega], v -> intersection[invar[IMAGE[SWAP]],
               invar[IMAGE[cross[inverse[SUCC], Id]]]], z → Z}]] // Reverse

Out[36]= subclass[intersection[invar[IMAGE[SWAP]], invar[IMAGE[cross[inverse[SUCC], Id]]]],
           union[image[S, set[Z]], P[complement[set[id[omega]]]]]] == True

In[37]:= % /. Equal → SetDelayed
```

Corollary.

```
In[38]:= SubstTest[subclass, dif[u, v], w,
           {u -> intersection[invar[IMAGE[SWAP]], invar[IMAGE[cross[inverse[SUCC], Id]]]],
            v -> P[complement[set[id[omega]]]], w -> image[S, set[Z]]}]

Out[38]= subclass[Z, hull[intersection[invar[IMAGE[SWAP]],
              invar[IMAGE[cross[inverse[SUCC], Id]]], set[id[omega]]]] == True

In[39]:= % /. Equal → SetDelayed
```

In the opposite direction, one has:

```
In[40]:= SubstTest[implies, member[z, w], subclass[A[w], z],
           {w -> intersection[invar[IMAGE[SWAP]], invar[IMAGE[cross[inverse[SUCC], Id]]],
              complement[P[complement[set[id[omega]]]]]], z → Z}]

Out[40]= subclass[hull[intersection[invar[IMAGE[SWAP]],
              invar[IMAGE[cross[inverse[SUCC], Id]]], set[id[omega]]], Z] == True

In[41]:= % /. Equal → SetDelayed
```

Combining these results, one finds:

```
In[42]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u -> hull[intersection[invar[IMAGE[SWAP]],
               invar[IMAGE[cross[inverse[SUCC], Id]]], set[id[omega]]], v -> Z}]

Out[42]= True == equal[Z, hull[intersection[invar[IMAGE[SWAP]],
              invar[IMAGE[cross[inverse[SUCC], Id]]], set[id[omega]]]]

In[43]:= hull[intersection[invar[IMAGE[SWAP]],
            invar[IMAGE[cross[inverse[SUCC], Id]]], set[id[omega]]] := Z
```

A similar formula holds for ordinary induction:

```
In[44]:= hull[invar[SUCC], set[0]]

Out[44]= omega
```