equipollence for map sets

Johan G. F. Belinfante
2006 July 4

```
In[1]:= SetDirectory["l:"]; << goedel83.02a; << tools.m

        :Package Title: goedel83.02a         2006 July 2 at 12:30 noon

        It is now:  2006 Jul 4 at 3:53

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 June 27

        weightlimit = 40
```

summary

The number of mappings from **x** to **y** depends only on the number of elements in **x** and the number of elements in **y**.

the double wrapper oopart[setpart[x]]

In the course of the derivation, the double wrapper **oopart[setpart[x]]** will need to be removed. In this section a surprisingly simple way to derive the needed rewrite rule for this is presented. One begins with this lemma:

```
In[2]:= SubstTest[implies, equal[x, funpart[setpart[y]]],
         equiv[equal[x, oopart[setpart[x]]], member[x, BIJ]], y → x]

Out[2]= or[equal[x, oopart[setpart[x]]], not[FUNCTION[x]],
          not[FUNCTION[inverse[x]]], not[member[x, V]]] == True

In[3]:= (% /. x → x_) /. Equal → SetDelayed
```

With this lemma in place, the GOEDEL program recognizes the truth of the needed rewrite rule:

```
In[4]:= equiv[equal[x, oopart[setpart[x]]],
         and[FUNCTION[x], FUNCTION[inverse[x]], member[x, V]]]

Out[4]= True
```

This rule is put into place:

```
In[5]:= equal[x_, oopart[setpart[x_]]] := and[FUNCTION[x], FUNCTION[inverse[x]], member[x, V]]
```

derivation

The idea in this section is to consider two map classes, with one-to-one correspondences between their domains and codomains. These one-to-one correspondences will be denoted by **oopart[u]** and **oopart[v]**, respectively. From each element **funpart[x]** of **map[domain[oopart[u]], domain[oopart[v]]]** one can construct an element **y** in **map[range[oopart[u]], range[oopart[v]]]** by the formula **y = composite[oopart[v], funpart[x], inverse[oopart[u]]]**. Lemma: **y** is a set.

```
In[6]:= implies[member[funpart[x], w],
          member[composite[oopart[v], funpart[x], inverse[oopart[u]]], V]] // AssertTest

Out[6]= or[member[composite[oopart[v], funpart[x], inverse[oopart[u]]], V],
          not[member[funpart[x], w]]] == True

In[7]:= (% /. {u → u_, v → v_, w → w_, x → x_}) /. Equal → SetDelayed
```

Lemma. The element **y** belongs to **map[range[oopart[u]], range[oopart[v]]]**.

```
In[8]:= Map[not, SubstTest[and, implies[p1, q1],
           implies[p1, p2], implies[and[p1, p2], q2], implies[p1, q3],
           implies[p1, q4], implies[and[q1, q2, q3, q4], q5], not[implies[p1, q5]],
           {p1 → and[member[funpart[x], V], equal[y, image[cross[oopart[u], oopart[v]],
                funpart[x]]], equal[domain[funpart[x]], domain[oopart[u]]],
              subclass[range[funpart[x]], domain[oopart[v]]]],
            p2 → equal[image[inverse[funpart[x]], domain[oopart[v]]], domain[funpart[x]]],
            q1 → FUNCTION[y], q2 → equal[domain[y], range[oopart[u]]],
            q3 → subclass[range[y], range[oopart[v]]], q4 → member[y, V],
            q5 → member[y, map[range[oopart[u]], range[oopart[v]]]]}]] /.
         y -> composite[oopart[v], funpart[x], inverse[oopart[u]]]

Out[8]= or[member[composite[oopart[v], funpart[x], inverse[oopart[u]]],
           map[range[oopart[u]], range[oopart[v]]]],
          not[equal[domain[funpart[x]], domain[oopart[u]]]], not[member[funpart[x], V]],
          not[subclass[range[funpart[x]], domain[oopart[v]]]]] == True

In[9]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Theorem. From each element of **map[domain[oopart[u]], domain[oopart[v]]]** one can construct an element in **map[range[oopart[u]], range[oopart[v]]]** by imaging with **cross[oopart[u], oopart[v]]**.

```
In[10]:= Map[not, SubstTest[and, implies[and[p0, p1, p2, p3], q5], implies[r0, p0],
           implies[r0, p2], implies[r0, p3], not[implies[and[r0, p1], q5]],
           {r0 → member[funpart[x], map[domain[oopart[u]], domain[oopart[v]]]],
            p0 → member[funpart[x], V],
            p1 → equal[y, image[cross[oopart[u], oopart[v]], funpart[x]]],
            p2 → equal[domain[funpart[x]], domain[oopart[u]]],
            p3 → subclass[range[funpart[x]], domain[oopart[v]]],
            p4 → equal[image[inverse[funpart[x]], domain[oopart[v]]], domain[funpart[x]]],
            q1 → FUNCTION[y], q2 → equal[domain[y], range[oopart[u]]],
            q3 → subclass[range[y], range[oopart[v]]], q4 → member[y, V],
            q5 → member[y, map[range[oopart[u]], range[oopart[v]]]]}]] /.
         y -> composite[oopart[v], funpart[x], inverse[oopart[u]]]

Out[10]= or[member[composite[oopart[v], funpart[x], inverse[oopart[u]]],
           map[range[oopart[u]], range[oopart[v]]]],
          not[member[funpart[x], map[domain[oopart[u]], domain[oopart[v]]]]]] == True

In[11]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Removing the **funpart** wrapper yields:

```
In[12]:= SubstTest[implies, equal[x, funpart[w]],
          or[member[composite[oopart[v]], x, inverse[oopart[u]]],
           map[range[oopart[u]], range[oopart[v]]]],
          not[member[x, map[domain[oopart[u]], domain[oopart[v]]]]], w → x]

Out[12]= or[member[composite[oopart[v], x, inverse[oopart[u]]],
           map[range[oopart[u]], range[oopart[v]]], not[FUNCTION[x]],
          not[member[x, map[domain[oopart[u]], domain[oopart[v]]]]]] == True

In[13]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Next the variable **x** is eliminated.

```
In[14]:= Map[equal[V, #] &, SubstTest[class, x,
           or[member[composite[oopart[v], x, inverse[oopart[u]]], y], not[FUNCTION[x]],
            not[member[x, z]]], {y -> map[range[oopart[u]], range[oopart[v]]],
            z -> map[domain[oopart[u]], domain[oopart[v]]]}]] // Reverse

Out[14]= subclass[image[IMAGE[cross[oopart[u], oopart[v]]],
           map[domain[oopart[u]], domain[oopart[v]]]],
          map[range[oopart[u]], range[oopart[v]]]] == True

In[15]:= (% /. {u → u_, v → v_}) /. Equal → SetDelayed
```

sharpening the inclusion to an equation

Replacing **u** and **v** by their inverses yields an inclusion in the other direction:

```
In[16]:= SubstTest[subclass,
          image[IMAGE[cross[oopart[s], oopart[t]]], map[domain[oopart[s]], domain[oopart[t]]]],
          map[range[oopart[s]], range[oopart[t]]], {s → inverse[u], t → inverse[v]}]

Out[16]= subclass[image[IMAGE[cross[inverse[oopart[u]], inverse[oopart[v]]]],
           map[range[oopart[u]], range[oopart[v]]]],
          map[domain[oopart[u]], domain[oopart[v]]]] == True

In[17]:= (% /. {u → u_, v → v_}) /. Equal → SetDelayed
```

These **IMAGE** functions are not quite inverses of each other, but instead satisfy:

```
In[18]:= composite[IMAGE[cross[oopart[u], oopart[v]]],
           IMAGE[cross[inverse[oopart[u]], inverse[oopart[v]]]]] // VSNormality

Out[18]= composite[IMAGE[cross[oopart[u], oopart[v]]],
           IMAGE[cross[inverse[oopart[u]], inverse[oopart[v]]]] ==
          IMAGE[id[cart[range[oopart[u]], range[oopart[v]]]]]

In[19]:= composite[IMAGE[cross[oopart[u_], oopart[v_]]],
           IMAGE[cross[inverse[oopart[u_]], inverse[oopart[v_]]]]] :=
          IMAGE[id[cart[range[oopart[u]], range[oopart[v]]]]]
```

Corollary.

```
In[21]:= ImageComp[IMAGE[cross[oopart[u], oopart[v]]],
           composite[IMAGE[cross[inverse[oopart[u]], inverse[oopart[v]]]],
            id[P[cart[range[oopart[u]], range[oopart[v]]]]]],
           map[range[oopart[u]], range[oopart[v]]]] // Reverse

Out[21]= image[IMAGE[cross[oopart[u], oopart[v]]],
           image[IMAGE[cross[inverse[oopart[u]], inverse[oopart[v]]]],
            map[range[oopart[u]], range[oopart[v]]]]] == map[range[oopart[u]], range[oopart[v]]]

In[22]:= image[IMAGE[cross[oopart[u_], oopart[v_]]],
           image[IMAGE[cross[inverse[oopart[u_]], inverse[oopart[v_]]]],
            map[range[oopart[u_]], range[oopart[v_]]]]] :=
          map[range[oopart[u]], range[oopart[v]]]
```

Theorem.

```
In[23]:= SubstTest[implies, subclass[s, t], subclass[image[r, s], image[r, t]],
          {r → IMAGE[cross[oopart[u], oopart[v]]],
           s -> image[IMAGE[cross[inverse[oopart[u]], inverse[oopart[v]]]],
             map[range[oopart[u]], range[oopart[v]]]],
           t -> map[domain[oopart[u]], domain[oopart[v]]]}]

Out[23]= subclass[map[range[oopart[u]], range[oopart[v]]],
           image[IMAGE[cross[oopart[u], oopart[v]]],
            map[domain[oopart[u]], domain[oopart[v]]]]] == True

In[24]:= (% /. {u → u_, v → v_}) /. Equal → SetDelayed
```

Combining the two inclusions yields an equation.

```
In[25]:= SubstTest[and, subclass[s, t], subclass[t, s],
          {s -> map[range[oopart[u]], range[oopart[v]]], t -> image[
             IMAGE[cross[oopart[u], oopart[v]]], map[domain[oopart[u]], domain[oopart[v]]]]}]

Out[25]= True == equal[image[IMAGE[cross[oopart[u], oopart[v]]],
            map[domain[oopart[u]], domain[oopart[v]]],
           map[range[oopart[u]], range[oopart[v]]]

In[26]:= image[IMAGE[cross[oopart[u_], oopart[v_]]],
           map[domain[oopart[u_]], domain[oopart[v_]]]] :=
          map[range[oopart[u]], range[oopart[v]]]
```

---

a one-to-one restriction of IMAGE[oopart[x]]

Although the function **IMAGE[oopart[x]]** need not be one-to-one, its restriction to the power class of **domain[oopart[x]]** is.

```
In[27]:= SubstTest[implies, subclass[composite[u, v], Id],
          FUNCTION[composite[inverse[v], id[domain[u]]], {u -> IMAGE[inverse[oopart[x]]],
           v → composite[IMAGE[oopart[x]], id[P[domain[oopart[x]]]]]}]

Out[27]= FUNCTION[composite[id[P[domain[oopart[x]]]], inverse[IMAGE[oopart[x]]]]] == True

In[28]:= FUNCTION[composite[id[P[domain[oopart[x_]]]], inverse[IMAGE[oopart[x_]]]]] := True
```

Since the cross product of one-to-one correspondences is one-to-one, the following corollary holds.

```
In[29]:= SubstTest[FUNCTION, composite[id[P[domain[oopart[z]]]], inverse[IMAGE[oopart[z]]]],
          z → cross[oopart[x], oopart[y]]]

Out[29]= FUNCTION[composite[id[P[cart[domain[oopart[x]], domain[oopart[y]]]]],
            inverse[IMAGE[cross[oopart[x], oopart[y]]]]]] == True

In[30]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

It follows that the two map classes under consideration are equipollent, provided **u** and **v** are sets. To insure thus, **setpart** wrappers can be introduced:

```
In[31]:= SubstTest[implies, and[member[w, BIJ], subclass[z, domain[w]]],
           member[pair[z, image[w, z]], Q],
           {w -> composite[IMAGE[cross[oopart[u], oopart[v]]],
              id[P[cart[domain[oopart[u]], domain[oopart[v]]]]]],
            z → map[domain[oopart[u]], domain[oopart[v]]]}] /. {u → setpart[x], v → setpart[y]}

Out[31]= member[pair[map[domain[oopart[setpart[x]]], domain[oopart[setpart[y]]]],
            map[range[oopart[setpart[x]]], range[oopart[setpart[y]]]], Q] == True

In[32]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

eliminating all variables

Removing the double wrappers **oopart[setpart[u]]** and **oopart[setpart[v]]** yields:

```
In[39]:= SubstTest[implies, and[equal[x, oopart[setpart[u]]], equal[y, oopart[setpart[v]]]],
          member[pair[pair[domain[x], domain[y]], pair[range[x], range[y]]],
           composite[inverse[MAP], Q, MAP]], {u → x, v → y}]

Out[39]= or[member[pair[map[domain[x], domain[y]], map[range[x], range[y]]], Q],
           not[FUNCTION[x]], not[FUNCTION[y]], not[FUNCTION[inverse[x]]],
           not[FUNCTION[inverse[y]]], not[member[x, V]], not[member[y, V]]] == True

In[40]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[33]:= member[pair[pair[u, v], w], composite[x, MAP]] // AssertTest // MapNotNot

Out[33]= member[pair[pair[u, v], w], composite[x, MAP]] ==
          and[member[u, V], member[v, V], member[w, V], member[pair[map[u, v], w], x]]

In[34]:= member[pair[pair[u_, v_], w_], composite[x_, MAP]] :=
          and[member[u, V], member[v, V], member[w, V], member[pair[map[u, v], w], x]]
```

Corollary.

```
In[36]:= SubstTest[member, pair[w, pair[u, v]],
           inverse[y], y → composite[inverse[x], MAP]] // MapNotNot

Out[36]= member[pair[w, pair[u, v]], composite[inverse[MAP], x]] ==
          and[member[u, V], member[v, V], member[w, V], member[pair[w, map[u, v]], x]]

In[37]:= member[pair[w_, pair[u_, v_]], composite[inverse[MAP], x_]] :=
          and[member[u, V], member[v, V], member[w, V], member[pair[w, map[u, v]], x]]
```

The variables **x** and **y** can now be elimiated:

```
In[41]:= Map[equal[0, composite[Id, complement[#]]] &,
           SubstTest[class, pair[x, y], implies[and[member[x, w], member[y, w]],
             member[pair[pair[domain[x], domain[y]], pair[range[x], range[y]]], z]],
            {w → BIJ, z -> composite[inverse[MAP], Q, MAP]}]] // Reverse

Out[41]= subclass[cart[BIJ, BIJ],
           fix[composite[cross[inverse[IMAGE[FIRST]], inverse[IMAGE[FIRST]]],
             inverse[MAP], Q, MAP, cross[IMAGE[SECOND], IMAGE[SECOND]]]]] == True

In[42]:= % /. Equal → SetDelayed
```

Theorem. A variable-free statement of the main theorem.

```
In[43]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u → id[cart[BIJ, BIJ]],
           v -> composite[cross[inverse[IMAGE[FIRST]], inverse[IMAGE[FIRST]]],
             inverse[MAP], Q, MAP, cross[IMAGE[SECOND], IMAGE[SECOND]]],
           w → cross[cross[IMAGE[SECOND], IMAGE[SECOND]], cross[IMAGE[FIRST], IMAGE[FIRST]]]}]

Out[43]= subclass[cross[Q, Q], composite[inverse[MAP], Q, MAP]] == True

In[44]:= subclass[cross[Q, Q], composite[inverse[MAP], Q, MAP]] := True
```

Corollary.

```
In[45]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u -> cross[Q, Q], v -> composite[inverse[MAP], Q, MAP], w → cross[MAP, MAP]}]

Out[45]= subclass[composite[MAP, cross[Q, Q], inverse[MAP]], Q] == True

In[46]:= subclass[composite[MAP, cross[Q, Q], inverse[MAP]], Q] := True
```

the main theorem

This main theorem is easier to understand when variables are reintroduced. Map classes are equipollent if their domains and codomains are equipollent.

```
In[47]:= SubstTest[implies, and[member[u, v], subclass[v, w]], member[u, w],
           {u → pair[pair[w, x], pair[y, z]],
            v -> cross[Q, Q], w -> composite[inverse[MAP], Q, MAP]}] // MapNotNot

Out[47]= or[member[pair[map[w, x], map[y, z]], Q],
           not[member[pair[w, y], Q]], not[member[pair[x, z], Q]]] == True

In[48]:= or[member[pair[map[w_, x_], map[y_, z_]], Q],
           not[member[pair[w_, y_], Q]], not[member[pair[x_, z_], Q]]] := True
```