image[inverse[S], map[x, y]]

Johan G. F. Belinfante
2007 October 16

```
In[1]:= SetDirectory["l:"]; << goedel98.14a; << tools.m

        :Package Title: goedel98.14a         2007 October 14 at 12:40 noon

        It is now:  2007 Oct 16 at 8:8

        Loading Simplification Rules

        TOOLS.M                                      Revised 2007 September 19

        weightlimit = 40
```

introduction

A formula is derived in this notebook for the class **image[inverse[S], map[x, y]]**. Any nonempty function **u** with domain **w** can be extended to any larger domain **x** without changing the range by simply taking the union of **u** with a constant function **cart[dif[x, w], set[v]]**, where **v** is any element of the range of **u**. This basic strategy will be exploited in this notebook to show that the mapping class **map[w, y]** is contained in the class **image[inverse[S], map[x, y]]** when **y** is not empty and **x** is a set. Using reification to remove the temporary variable **w** yields a crucial lower bound from which the desired result follows. One can pursue the use of reification still further to obtain a completely variable-free result.

an upper bound

This upper bound is easily derived. This result will later be replaced by a more precise equation.

```
In[2]:= subclass[image[inverse[S], map[x, y]], FUNS] // AssertTest
Out[2]= subclass[image[inverse[S], map[x, y]], FUNS] == True

In[3]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

disjoint domains

The union of two functions with disjoint domains is a function. The following lemma can be obtained using **AssertTest**.

```
In[4]:= or[member[union[u, v], map[union[domain[u], domain[v]], z]],
          not[equal[0, intersection[domain[u], domain[v]]]],
          not[member[u, map[domain[u], z]]], not[member[v, map[domain[v], z]]]] // AssertTest

Out[4]= or[member[union[u, v], map[union[domain[u], domain[v]], z]],
          not[equal[0, intersection[domain[u], domain[v]]]],
          not[member[u, map[domain[u], z]]], not[member[v, map[domain[v], z]]] == True

In[5]:= (% /. {u → u_, v → v_, z → z_}) /. Equal → SetDelayed
```

Theorem. One can join mappings with disjoint domains.

```
In[6]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p1, p3], implies[and[p1, p2, p3], p4], not[implies[p1, p4]],
           {p1 → and[member[u, map[x, z]], member[v, map[y, z]], disjoint[x, y]],
            p2 → equal[x, domain[u]], p3 → equal[y, domain[v]],
            p4 -> member[union[u, v], map[union[x, y], z]]}]] // Reverse

Out[6]= or[member[union[u, v], map[union[x, y], z]], not[equal[0, intersection[x, y]]],
          not[member[u, map[x, z]]], not[member[v, map[y, z]]] == True

In[7]:= or[member[union[u_, v_], map[union[x_, y_], z_]], not[equal[0, intersection[x_, y_]]],
          not[member[u_, map[x_, z_]]], not[member[v_, map[y_, z_]]] := True
```

Corollary. In particular, one can extend functions using constant functions.

```
In[8]:= Map[implies[member[x, V], #] &,
         SubstTest[implies, equal[x, union[w, t]], or[member[union[u, q], map[x, y]],
             not[equal[0, intersection[w, t]]], not[member[u, map[w, y]]],
             not[member[q, map[t, y]]]], {q -> cart[intersection[x, complement[w]], set[v]],
             t → dif[x, w]}] // Reverse // MapNotNot]

Out[8]= or[member[union[u, cart[intersection[x, complement[w]], set[v]]], map[x, y]],
          not[member[u, map[w, y]]], not[member[v, y]],
          not[member[x, V]], not[subclass[w, x]]] == True

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

Lemma.

```
In[10]:= SubstTest[implies, and[subclass[x, t], member[t, z]],
           member[x, image[inverse[S], z]], t -> union[x, y]] // Reverse

Out[10]= or[member[x, image[inverse[S], z]], not[member[union[x, y], z]]] == True

In[11]:= or[member[x_, image[inverse[S], z_]], not[member[union[x_, y_], z_]]] := True
```

Theorem.

```
In[12]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 → and[member[u, map[w, y]], member[v, y], member[x, V], subclass[w, x]],
            p2 -> member[union[u, cart[intersection[x, complement[w]], set[v]]], map[x, y]],
            p3 → member[u, image[inverse[S], map[x, y]]]}]] // Reverse

Out[12]= or[member[u, image[inverse[S], map[x, y]]], not[member[u, map[w, y]]],
           not[member[v, y]], not[member[x, V]], not[subclass[w, x]]] == True

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

Eliminating the variables **u** and **v** is easy:

```
In[14]:= Map[empty[composite[Id, complement[#]]] &,
          SubstTest[class, pair[u, v], or[member[u, t], not[member[u, map[w, y]]],
            not[member[v, y]], not[member[x, V]], not[subclass[w, x]]],
           {t -> image[inverse[S], map[x, y]], z → map[w, y]}]]

Out[14]= or[equal[0, y], not[member[x, V]], not[subclass[w, x]],
           subclass[map[w, y], image[inverse[S], map[x, y]]]] == True

In[15]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

The variable **w** can be eliminated using **reify** as follows. (This takes a while.)

```
In[16]:= Map[empty[composite[Id, complement[#]]] &, SubstTest[reify, w,
           union[complement[image[V, y]], complement[image[V, set[x]]], complement[map[w, y]],
            image[V, intersection[w, complement[x]]], z], z → image[inverse[S], map[x, y]]]]

Out[16]= or[equal[0, y], not[member[x, V]],
           subclass[intersection[FUNS, P[cart[x, y]]], image[inverse[S], map[x, y]]]] == True

In[17]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The inclusion can be sharpened to an equation.

```
In[18]:= SubstTest[and, implies[p, subclass[u, v]], subclass[v, u],
          {p → and[member[x, V], not[empty[y]]],
           u -> intersection[FUNS, P[cart[x, y]]], v -> image[inverse[S], map[x, y]]}]

Out[18]= or[equal[0, y], equal[image[inverse[S], map[x, y]], intersection[FUNS, P[cart[x, y]]]],
           not[member[x, V]]] == True

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

In the reverse direction, one has:

```
In[20]:= SubstTest[implies, equal[u, v], equal[U[u], U[v]],
           {u -> image[inverse[S], map[x, y]], v -> intersection[FUNS, P[cart[x, y]]]}] // Reverse

Out[20]= or[equal[0, y], member[x, V],
           not[equal[image[inverse[S], map[x, y]], intersection[FUNS, P[cart[x, y]]]]]] == True

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

One literal is redundant and can be removed:

```
In[22]:= SubstTest[and, or[p, q], implies[p, q], {p → not[empty[y]], q :> or[member[x, V],
           not[equal[image[inverse[S], map[x, y]], intersection[FUNS, P[cart[x, y]]]]]]}]

Out[22]= or[member[x, V],
          not[equal[image[inverse[S], map[x, y]], intersection[FUNS, P[cart[x, y]]]]] == True

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

Combining these results, one obtains the following temporary rewrite rule.

```
In[24]:= equiv[equal[image[inverse[S], map[x, y]], intersection[FUNS, P[cart[x, y]]]],
           and[member[x, V], implies[empty[y], empty[x]]]] // not // not

Out[24]= True

In[25]:= equal[image[inverse[S], map[x_, y_]], intersection[FUNS, P[cart[x_, y_]]]] :=
          or[and[member[x, V], not[equal[0, y]]], equal[0, x]]
```

Lemma.

```
In[26]:= equal[intersection[FUNS, complement[image[V, x]], P[cart[x, y]]],
          intersection[complement[image[V, x]], set[0]]]

Out[26]= True

In[27]:= intersection[FUNS, complement[image[V, x_]], P[cart[x_, y_]]] :=
          intersection[complement[image[V, x]], set[0]]
```

An even better result can now be derived:

```
In[28]:= equal[image[inverse[S], map[x, y]],
           union[intersection[complement[image[V, x]], set[0]],
            intersection[FUNS, image[V, y], image[V, set[x]], P[cart[x, y]]]]] // AssertTest

Out[28]= equal[image[inverse[S], map[x, y]],
          union[intersection[complement[image[V, x]], set[0]],
           intersection[FUNS, image[V, y], image[V, set[x]], P[cart[x, y]]]]] == True

In[29]:= image[inverse[S], map[x_, y_]] := union[intersection[complement[image[V, x]], set[0]],
           intersection[FUNS, image[V, y], image[V, set[x]], P[cart[x, y]]]]
```

eliminating the remaining variables

Lemma.

```
In[30]:= composite[ADJOIN[set[0]], IMAGE[id[FUNS]], POWER,
           CART, id[cart[set[0], complement[set[0]]]]] // ReifTriNormality

Out[30]= composite[ADJOIN[set[0]], IMAGE[id[FUNS]],
           POWER, CART, id[cart[set[0], complement[set[0]]]]] ==
         cart[cart[set[0], complement[set[0]]], set[set[0]]]

In[31]:= composite[ADJOIN[set[0]], IMAGE[id[FUNS]],
           POWER, CART, id[cart[set[0], complement[set[0]]]]] :=
         cart[cart[set[0], complement[set[0]]], set[set[0]]]

In[32]:= Map[composite[VERTSECT[#], id[cart[V, V]]] &,
          SubstTest[reify, x, image[z, map[first[x], second[x]]], z → inverse[S]]]

Out[32]= composite[IMAGE[inverse[S]], MAP] == union[
           cart[cart[complement[set[0]], set[0]], set[0]], cart[cart[set[0], V], set[set[0]]],
           composite[IMAGE[id[FUNS]], POWER, id[complement[set[0]]], CART]]

In[33]:= composite[IMAGE[inverse[S]], MAP] := union[
           cart[cart[complement[set[0]], set[0]], set[0]], cart[cart[set[0], V], set[set[0]]],
           composite[IMAGE[id[FUNS]], POWER, id[complement[set[0]]], CART]]
```

RESTRICT corollary

The function **IMAGE[inverse[S]]** may be replaced with **IMAGE[RESTRICT]**, if desired. The following lemma provides the key idea.

```
In[34]:= Map[VERTSECT, Assoc[RESTRICT, id[FUNS], inverse[E]]]

Out[34]= composite[IMAGE[RESTRICT], IMAGE[id[FUNS]]] ==
          composite[IMAGE[inverse[S]], IMAGE[id[FUNS]]]

In[35]:= composite[IMAGE[RESTRICT], IMAGE[id[FUNS]]] :=
          composite[IMAGE[inverse[S]], IMAGE[id[FUNS]]]
```

Corollary.

```
In[36]:= Assoc[IMAGE[RESTRICT], IMAGE[id[FUNS]], MAP]

Out[36]= composite[IMAGE[RESTRICT], MAP] == union[
           cart[cart[complement[set[0]], set[0]], set[0]], cart[cart[set[0], V], set[set[0]]],
           composite[IMAGE[id[FUNS]], POWER, id[complement[set[0]]], CART]]

In[37]:= composite[IMAGE[RESTRICT], MAP] := union[
           cart[cart[complement[set[0]], set[0]], set[0]], cart[cart[set[0], V], set[set[0]]],
           composite[IMAGE[id[FUNS]], POWER, id[complement[set[0]]], CART]]
```