bijective transforms of partial orders

Johan G. F. Belinfante
2005 January 17

In[1]:= **SetDirectory["i:"]; << goedel65.16a; << tools.m**

```
:Package Title: goedel65.16a           2005 January 16 at 8:30 p.m.

It is now:  2005 Jan 17 at 9:52

Loading Simplification Rules

TOOLS.M                       Revised 2005 January 7

weightlimit = 40
```

---

summary

By renaming the elements of a partially ordered set one obtains another. For reflexive and transitive relations a more general result holds. The reflexive and transitive properties are preserved by transformations using arbitrary functions, not just bijections:

In[2]:= **implies[and[FUNCTION[x], REFLEXIVE[y]],
REFLEXIVE[composite[inverse[x], y, x]]]**

Out[2]= True

In[3]:= **implies[and[FUNCTION[x], TRANSITIVE[y]],
TRANSITIVE[composite[inverse[x], y, x]]]**

Out[3]= True

The antisymmetric property, on the other hand, need not be preserved under such transformations unless **x** is one-to-one, and the same is true for partial ordering relations. The results about partial orders will be derived from corresponding facts about reflexive, antisymmetric and transitive relations. Some lemmas concerning reflexive and antisymmetric relations are needed to do this. At the end of the notebook a simple counterexample is presented which shows that one cannot omit the one-to-one hypthesis on **x** for the theorems derived about antisymmetric relations and partial orderings.

lemma about RFX

In this section a needed result is derived for the class **RFX** of reflexive relations. The main ideas are just that the class **P[Id]** of identity functions a subclass of the class **BIJ** of bijections, and the latter is a subclass of the class **image[INVERSE, FUNS]** of inverse functions. The technique is to use the monotonicity of imaging with the following function:

```
In[4]:= abstract[x, image[IMG, cart[image[CROSS, id[x]], RFX]]]

Out[4]= composite[IMG, id[cart[V, RFX]], inverse[FIRST], CROSS, DUP]
```

This yields two inclusions in opposite directions:

```
In[5]:= SubstTest[implies, subclass[u, v],
          subclass[image[w, u], image[w, v]], {u → P[Id], v → BIJ,
           w -> composite[IMG, id[cart[V, RFX]], inverse[FIRST], CROSS, DUP]}]

Out[5]= subclass[RFX, image[IMG, cart[image[CROSS, id[BIJ]], RFX]]] == True

In[6]:= % /. Equal → SetDelayed

In[7]:= SubstTest[implies, subclass[u, v],
          subclass[image[w, u], image[w, v]], {u → BIJ, v → image[INVERSE, FUNS],
           w -> composite[IMG, id[cart[V, RFX]], inverse[FIRST], CROSS, DUP]}]

Out[7]= subclass[image[IMG, cart[image[CROSS, id[BIJ]], RFX]], RFX] == True

In[8]:= % /. Equal → SetDelayed
```

The two inclusions are combined into an equation:

```
In[9]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> image[IMG, cart[image[CROSS, id[BIJ]], RFX]], v -> RFX}]

Out[9]= True == equal[RFX, image[IMG, cart[image[CROSS, id[BIJ]], RFX]]]

In[10]:= image[IMG, cart[image[CROSS, id[BIJ]], RFX]] := RFX
```

restrictions of antisymmetric relations

Restrictions of antisymmetric relations are studied in this section. A double-negation lemma prepares for the removal of variables.

```
In[11]:= (implies[member[y, z], member[restrict[y, x, x], z]] /. z → ANTISYM) //
          NotNotTest

Out[11]= or[and[member[intersection[x, image[y, x]], V],
            member[intersection[x, image[inverse[y], x]], V],
            subclass[composite[id[x], intersection[y, inverse[y]], id[x]], Id]],
          not[member[y, V]], not[subclass[y, cart[V, V]]],
          not[subclass[intersection[y, inverse[y]], Id]]] == True

In[12]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The variable **y** is eliminated first.

```
In[13]:= Map[equal[V, #] &, SubstTest[class, y,
            implies[member[y, z], member[restrict[y, x, x], z]], z → ANTISYM]] // Reverse

Out[13]= subclass[image[IMAGE[id[cart[x, x]]], ANTISYM], ANTISYM] == True

In[14]:= subclass[image[IMAGE[id[cart[x_, x_]]], ANTISYM], ANTISYM] := True
```

Reification is used to remove the variable **x**.

```
In[15]:= Map[equal[0, #] &, SubstTest[reify, x,
            dif[image[IMAGE[id[cart[x, x]]], y], y], y → ANTISYM]] // Reverse

Out[15]= subclass[image[IMG, cart[image[IMAGE[DUP], image[CART, Id]], ANTISYM]],
          ANTISYM] == True

In[16]:= % /. Equal → SetDelayed
```

The result can be rewritten in a simpler form:

```
In[17]:= Map[subclass[#, ANTISYM] &,
          ImageComp[IMG, cross[IDP, Id], cart[image[CART, Id], ANTISYM]]]

Out[17]= subclass[image[CAP, cart[image[CART, Id], ANTISYM]], ANTISYM] == True

In[18]:= % /. Equal → SetDelayed
```

The symmetry of **CAP** allows one to flip the arguments of **cart**.

```
In[19]:= Map[subclass[#, ANTISYM] &,
          ImageComp[CAP, SWAP, cart[image[CART, Id], ANTISYM]]] // Reverse

Out[19]= subclass[image[CAP, cart[ANTISYM, image[CART, Id]]], ANTISYM] == True

In[20]:= % /. Equal → SetDelayed
```

The reverse inclusion also holds:

```
In[21]:= SubstTest[implies, subclass[u, v],
          subclass[image[u, w], image[v, w]], {u -> id[P[cart[V, V]]], v ->
            composite[CAP, id[cart[V, image[CART, Id]]], inverse[FIRST]], w → ANTISYM}]

Out[21]= subclass[ANTISYM, image[CAP, cart[ANTISYM, image[CART, Id]]]] == True

In[22]:= % /. Equal → SetDelayed
```

The two inclusions are combined into an equation.

```
In[23]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> ANTISYM, v -> image[CAP, cart[ANTISYM, image[CART, Id]]]}]

Out[23]= True == equal[ANTISYM, image[CAP, cart[ANTISYM, image[CART, Id]]]]

In[24]:= image[CAP, cart[ANTISYM, image[CART, Id]]] := ANTISYM
```

There is a companion result with the **cart** arguments interchanged:

```
In[25]:= ImageComp[CAP, SWAP, cart[image[CART, Id], ANTISYM]]

Out[25]= image[CAP, cart[image[CART, Id], ANTISYM]] == ANTISYM

In[26]:= image[CAP, cart[image[CART, Id], ANTISYM]] := ANTISYM
```

---

lemma about ANTISYM

In this section transforms of antisymmetric relations under bijections are studied. In the first step the **oopart** wrapper is used to construct a generic bijection.

```
In[27]:= SubstTest[implies, subclass[u, v],
          subclass[image[inverse[w], u], image[inverse[w], v]],
          {u -> intersection[y, inverse[y]], v → Id, w → cross[oopart[x], oopart[x]]}]

Out[27]= or[not[subclass[intersection[y, inverse[y]], Id]],
           subclass[composite[inverse[oopart[x]], intersection[y, inverse[y]],
             oopart[x]], id[domain[oopart[x]]]]] == True

In[28]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

```
In[29]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
           not[implies[p1, p3]], {p1 -> subclass[intersection[y, inverse[y]], Id],
            p2 -> subclass[composite[inverse[oopart[x]],
               intersection[y, inverse[y]], oopart[x]], id[domain[oopart[x]]]],
            p3 -> subclass[composite[inverse[oopart[x]],
               intersection[y, inverse[y]], oopart[x]], Id]}]]

Out[29]= or[not[subclass[intersection[y, inverse[y]], Id]],
           subclass[composite[inverse[oopart[x]],
             intersection[y, inverse[y]], oopart[x]], Id]] == True

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

The **oopart** wrapper is removed and replaced with a pair of **FUNCTION** literals

```
In[31]:= SubstTest[implies, and[equal[z, oopart[x]], ANTISYMMETRIC[y]],
          ANTISYMMETRIC[composite[inverse[z], y, z]], z → x]

Out[31]= or[not[FUNCTION[x]], not[FUNCTION[inverse[x]]], not[subclass[y, cart[V, V]]],
           not[subclass[intersection[y, inverse[y]], Id]],
           subclass[intersection[composite[inverse[x], y, x],
             composite[inverse[x], inverse[y], x]], Id]] == True

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

A sethood lemma is needed. This rewrite rule appears to be generally useful, and can be made permanent.

```
In[33]:= Map[implies[member[x, z], #] &, SubstTest[implies,
           equal[z, setpart[x]], member[composite[inverse[z], y, z], V], z → x]]

Out[33]= or[member[composite[inverse[x], y, x], V], not[member[x, z]]] == True

In[34]:= or[member[composite[inverse[x_], y_, x_], V], not[member[x_, z_]]] := True
```

Again a double-negation lemma is needed to prepare for the removal of variables:

```
In[35]:= implies[and[member[x, BIJ], member[y, ANTISYM]],
           member[composite[inverse[x], y, x], ANTISYM]] // NotNotTest

Out[35]= or[and[member[composite[inverse[x], y, x], V], subclass[intersection[
              composite[inverse[x], y, x], composite[inverse[x], inverse[y], x]], Id]],
           not[FUNCTION[x]], not[FUNCTION[inverse[x]]], not[member[x, V]],
           not[member[y, V]], not[subclass[y, cart[V, V]]],
           not[subclass[intersection[y, inverse[y]], Id]]] == True

In[36]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

All the variables can now be removed at once:

```
In[37]:= Map[equal[0, composite[complement[#], id[cart[V, V]]]] &,
           SubstTest[class, pair[pair[x, y], z],
            implies[and[member[pair[pair[x, x], z], w], member[x, u], member[y, v]],
             member[image[inverse[z], y], v]],
            {u → BIJ, v → ANTISYM, w → CROSS}]] // Reverse

Out[37]= subclass[image[IMG, cart[image[CROSS, id[BIJ]], ANTISYM]], ANTISYM] == True

In[38]:= % /. Equal → SetDelayed
```

In the reverse direction, a lemma is needed:

```
In[39]:= ImageComp[IMG, cross[IDP, Id], cart[image[CART, Id], ANTISYM]] // Reverse

Out[39]= image[IMG, cart[image[IMAGE[DUP], image[CART, Id]], ANTISYM]] == ANTISYM

In[40]:= % /. Equal → SetDelayed
```

The inclusion in the opposite direction follows:

```
In[41]:= SubstTest[implies, subclass[u, v],
          subclass[image[w, u], image[w, v]], {u → P[Id], v → BIJ,
           w -> composite[IMG, id[cart[V, ANTISYM]], inverse[FIRST], CROSS, DUP]}]

Out[41]= subclass[ANTISYM, image[IMG, cart[image[CROSS, id[BIJ]], ANTISYM]]] == True

In[42]:= % /. Equal → SetDelayed
```

The two inclusions are combined into an equation:

```
In[43]:= SubstTest[and, subclass[u, v], subclass[v, u], {u → ANTISYM,
            v -> image[IMG, cart[image[CROSS, id[BIJ]], ANTISYM]]}] // Reverse

Out[43]= equal[ANTISYM, image[IMG, cart[image[CROSS, id[BIJ]], ANTISYM]]] == True

In[44]:= image[IMG, cart[image[CROSS, id[BIJ]], ANTISYM]] := ANTISYM
```

---

corollary for partial order

Lemma.

```
In[45]:= ImageComp[IMG, cross[IDP, Id], cart[image[CART, Id], PO]] // Reverse

Out[45]= image[IMG, cart[image[IMAGE[DUP], image[CART, Id]], PO]] == PO

In[46]:= % /. Equal → SetDelayed
```

```
In[47]:= SubstTest[implies, subclass[u, v],
          subclass[image[w, u], image[w, v]], {u → P[Id], v → BIJ,
           w -> composite[IMG, id[cart[V, PO]], inverse[FIRST], CROSS, DUP]}]

Out[47]= subclass[PO, image[IMG, cart[image[CROSS, id[BIJ]], PO]]] == True

In[48]:= % /. Equal → SetDelayed
```

The reverse inclusion holds for reflexive, antisymmetric and transitive relations, and therefore for partial order relations:

```
In[49]:= SubstTest[subclass, image[w, intersection[x, y, z]],
          intersection[image[w, x], image[w, y], image[w, z]],
          {w -> composite[IMG, id[cart[image[CROSS, id[BIJ]], V]], inverse[SECOND]],
           x → RFX, y → ANTISYM, z → TRV}]

Out[49]= subclass[image[IMG, cart[image[CROSS, id[BIJ]], PO]], PO] == True

In[50]:= % /. Equal → SetDelayed
```

These two inclusions are combined into an equation:

```
In[51]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> image[IMG, cart[image[CROSS, id[BIJ]], PO]], v → PO}]

Out[51]= True == equal[PO, image[IMG, cart[image[CROSS, id[BIJ]], PO]]]

In[52]:= image[IMG, cart[image[CROSS, id[BIJ]], PO]] := PO
```

a counterexample

In this section, an explicit couterexample is given to show that one cannot replace the **ONEONE** hypothesis by **FUNCTION**. The function in the counterexample is the constant function which takes **0** and **1** to **0**.

```
In[53]:= FUNCTION[cart[succ[set[0]], set[0]]]

Out[53]= True
```

The antisymmetric relation in this counterexample is the less-than-or-equal relation for the set of numbers **{0, 1}**.

```
In[54]:= ANTISYMMETRIC[composite[id[succ[set[0]]], S, id[succ[set[0]]]]]

Out[54]= True
```

The transformed relation is a cartesian square, which is not antisymmetric.

```
In[55]:= (composite[inverse[x], y, x] /. {x -> cart[succ[set[0]], set[0]],
            y → composite[id[succ[set[0]]], S, id[succ[set[0]]]]})

Out[55]= cart[succ[set[0]], succ[set[0]]]

In[56]:= cart[succ[set[0]], succ[set[0]]] // ANTISYMMETRIC

Out[56]= False
```

A cartesian square **cart[x,x]** is a partial order only when **x** has less than 2 members.

```
In[57]:= PARTIALORDER[cart[x, x]] // AssertTest

Out[57]= PARTIALORDER[cart[x, x]] == or[equal[0, x], member[x, range[SINGLETON]]]

In[58]:= PARTIALORDER[cart[x_, x_]] := or[equal[0, x], member[x, range[SINGLETON]]]
```

It follows that the same counterexample works for partial order.

```
In[59]:= implies[and[FUNCTION[x], PARTIALORDER[y]],
           PARTIALORDER[composite[inverse[x], y, x]]] /.
          {x -> cart[succ[set[0]], set[0]],
           y → composite[id[succ[set[0]]], S, id[succ[set[0]]]]}

Out[59]= False
```