partial orderings

Johan G. F. Belinfante
2003 September 24

```
In[1]:= << goedel52.s96; << tools.m

        :Package Title: goedel52.s96      2003 September 21 at 8:45 p.m.

        It is now:  2003 Sep 29 at 13:20

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 August 9

        weightlimit = 40
```

summary

In this notebook some facts are derived about the class **PO** of all partial orderings. Recall the basic definitions:

```
In[2]:= PARTIALORDER[x]

Out[2]= and[subclass[x, cart[fix[x], fix[x]]],
          subclass[intersection[x, inverse[x]], Id], TRANSITIVE[x]]

In[3]:= class[x, PARTIALORDER[x]]

Out[3]= PO
```

The class of partial orderings is the intersection of the classes of reflexive relations, antisymmetric relations and transitive relations:

```
In[4]:= intersection[RFX, ANTISYM, TRV]

Out[4]= PO
```

some obvious facts

In this section some obvious facts are derived that will be needed in the sequel.

```
In[5]:= subclass[PO, RFX] // AssertTest

Out[5]= subclass[PO, RFX] == True

In[6]:= subclass[PO, RFX] := True

In[7]:= subclass[PO, ANTISYM] // AssertTest

Out[7]= subclass[PO, ANTISYM] == True
```

```
In[8]:= subclass[PO, ANTISYM] := True
```

In the case of **TRV** a different approach is required:

```
In[9]:= SubstTest[subclass, intersection[u, v], v, {u -> intersection[RFX, ANTISYM], v -> TRV}]

Out[9]= subclass[PO, TRV] == True

In[10]:= subclass[PO, TRV] := True
```

cores

It will be shown below that the functions **CORE[PO]** and **CORE[RFX]** are equal. This fact allows one to establish some elementary properties about the class **PO**.

```
In[11]:= core[x_, y_] := U[intersection[x, P[y]]]
```

The monotonicity of cores yields an inclusion in one direction:

```
In[12]:= SubstTest[implies, subclass[u, v], subclass[core[u, x], core[v, x]], {u -> PO, v -> RFX}]

Out[12]= subclass[U[intersection[PO, P[x]]], cart[fix[x], fix[x]]] == True

In[13]:= subclass[U[intersection[PO, P[x_]]], cart[fix[x_], fix[x_]]] := True

In[14]:= subclass[core[PO, x], core[RFX, x]]

Out[14]= True
```

The reverse inclusion will be derived shortly.

el

In this section an arbitrary reflexive relation is written as the union of its partially ordered subsets. For this purpose it is useful to introduce the smallest partial order relations, which are three−point sets shaped like the letter **L**. In the digraph interpretation, these are graphs with a single directed edge with self−loops at each end.

```
In[15]:= member[union[cart[singleton[x], singleton[y]], id[pairset[x, y]]], PO]

Out[15]= True
```

A temporary name for these partial order relations is introduced:

```
In[16]:= el[x_, y_] := union[cart[singleton[x], singleton[y]], id[pairset[x, y]]]

In[17]:= (SubstTest[implies, and[member[w, x], subclass[x, y]], member[w, y],
            {w -> pair[u, v], y -> cart[fix[x], fix[x]]}] /. {u -> u_, v -> v_, x -> x_}) /.
          Equal -> SetDelayed
```

```
In[18]:= Map[implies[and[REFLEXIVE[x], member[pair[u, v], x]], #] &,
          subclass[el[u, v], x] // AssertTest]

Out[18]= or[not[member[pair[u, v], x]],
           not[subclass[x, cart[fix[x], fix[x]]]], subclass[pairset[u, v], fix[x]]] == True

In[19]:= (% /. {u -> u_, v -> v_, x -> x_}) /. Equal -> SetDelayed
```

From this one can deduce an inclusion of cores:

```
In[20]:= Map[equal[0, composite[Id, complement[class[pair[u, v], #]]]] &,
          Map[or[not[member[v, V]], #] &,
           SubstTest[implies, and[member[w, y], member[y, z]], member[w, U[z]],
             {w -> pair[u, v], y -> el[u, v], z -> intersection[PO, P[x]]}]]]

Out[20]= subclass[composite[id[fix[x]], x, id[fix[x]]], U[intersection[PO, P[x]]]] == True

In[21]:= (% /. x -> x_) /. Equal -> SetDelayed
```

Putting the two inclusions together yields an equation:

```
In[22]:= SubstTest[and, subclass[u, v],
           subclass[v, u], {u → core[PO, x], v → core[RFX, x]}] // Reverse

Out[22]= equal[composite[id[fix[x]], x, id[fix[x]]], U[intersection[PO, P[x]]]] == True

In[23]:= U[intersection[PO, P[x_]]] := composite[id[fix[x]], x, id[fix[x]]]
```

Reformulation:

```
In[24]:= core[PO, x] == core[RFX, x]

Out[24]= True
```

functional formulation of the core results

There is a variable−free formulation of the result derived in the last section. In the course of deriving it, one discovers also a formula for the **Uclosure** of **PO**.

```
In[25]:= symdif[CORE[PO], CORE[RFX]] // VSNormality

Out[25]= union[intersection[complement[CORE[PO]], CORE[RFX]],
           intersection[complement[CORE[RFX]], CORE[PO]]] == 0

In[26]:= union[intersection[complement[CORE[PO]], CORE[RFX]],
           intersection[complement[CORE[RFX]], CORE[PO]]] := 0

In[27]:= SubstTest[equal, 0, symdif[u, v], {u -> CORE[PO], v -> CORE[RFX]}]

Out[27]= True == equal[RFX, Uclosure[PO]]

In[28]:= Uclosure[PO] := RFX

In[29]:= equal[CORE[PO], CORE[RFX]]

Out[29]= True
```

```
In[30]:= CORE[PO] := CORE[RFX]
```

Corollary:

```
In[31]:= SubstTest[U, Uclosure[x], x -> PO] // Reverse

Out[31]= U[PO] == cart[V, V]

In[32]:= U[PO] := cart[V, V]
```

an unrelated fact

The following additional fact about **CORE[RFX]** is unrelated to the main content of this notebook, and is included only for completeness.

```
In[33]:= SubstTest[composite, CORE[invar[x]], CAP,
             x -> union[composite[DUP, union[FIRST, SECOND]], cart[complement[cart[V, V]], V]]]

Out[33]= composite[CORE[RFX], CAP] == composite[CAP, cross[CORE[RFX], CORE[RFX]]]

In[34]:= composite[CORE[RFX], CAP] := composite[CAP, cross[CORE[RFX], CORE[RFX]]]
```

hulls

The case of hulls is more interesting than that of cores.

```
In[35]:= hull[x_, y_] := A[intersection[x, image[S, singleton[y]]]]
```

A general property of hulls:

```
In[36]:= SubstTest[implies, member[z, w], subclass[A[w], z],
         w -> intersection[x, image[S, singleton[y]]]]

Out[36]= or[not[member[z, x]], not[subclass[y, z]],
           subclass[A[intersection[x, image[S, singleton[y]]]], z]] == True

In[37]:= or[not[member[z_, x_]], not[subclass[y_, z_]],
           subclass[A[intersection[x_, image[S, singleton[y_]]]], z_]] := True
```

Restatement:

```
In[38]:= implies[and[subclass[y, z], member[z, x]], subclass[hull[x, y], z]]

Out[38]= True
```

reflexive hulls

```
In[39]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 -> subclass[x, cart[V, V]], p2 -> subclass[x, cart[domain[x], range[x]]],
             p3 -> REFLEXIVE[union[x, id[union[domain[x], range[x]]]]]}]]

Out[39]= or[not[subclass[x, cart[V, V]]],
           subclass[x, cart[union[domain[x], range[x]], union[domain[x], range[x]]]]] == True

In[40]:= (% /. x -> x_) /. Equal -> SetDelayed

In[41]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[p3, p4], implies[p3, p5], not[implies[and[p1, p2], and[p4, p5]]],
             {p1 -> subclass[x, y], p2 -> REFLEXIVE[y], p3 -> subclass[x, cart[fix[y], fix[y]]],
               p4 -> subclass[domain[x], fix[y]], p5 -> subclass[range[x], fix[y]]}]]

Out[41]= or[and[subclass[domain[x], fix[y]], subclass[range[x], fix[y]]],
           not[subclass[x, y]], not[subclass[y, cart[fix[y], fix[y]]]]] == True

In[42]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed

In[43]:= Map[complement,
           SubstTest[class, y, implies[and[subclass[x, y], REFLEXIVE[y]], subclass[z, y]],
             z -> union[x, id[union[domain[x], range[x]]]]] // Reverse]

Out[43]= intersection[RFX,
           complement[image[S, singleton[union[x, id[union[domain[x], range[x]]]]]],
           image[S, singleton[x]]] == 0

In[44]:= Map[equal[V, #] &,
           SubstTest[class, y, implies[and[subclass[x, y], REFLEXIVE[y]], subclass[z, y]],
             z -> union[x, id[union[domain[x], range[x]]]]] // Reverse]

Out[44]= and[subclass[domain[x], fix[A[intersection[RFX, image[S, singleton[x]]]]]],
           subclass[range[x], fix[A[intersection[RFX, image[S, singleton[x]]]]]]] == True

In[45]:= (% /. x -> x_) /. Equal -> SetDelayed

In[46]:= SubstTest[implies, and[member[y, z], subclass[x, y]],
           subclass[A[intersection[z, image[S, singleton[x]]]], y],
             {y -> union[x, id[union[domain[x], range[x]]]], z -> RFX}]

Out[46]= or[not[member[x, V]],
           not[subclass[x, cart[union[domain[x], range[x]], union[domain[x], range[x]]]]],
           subclass[A[intersection[RFX, image[S, singleton[x]]]],
             union[x, id[union[domain[x], range[x]]]]] == True

In[47]:= (% /. x -> x_) /. Equal -> SetDelayed

In[48]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 -> member[x, P[cart[V, V]]], p2 -> and[member[x, V],
             subclass[x, cart[union[domain[x], range[x]], union[domain[x], range[x]]]]],
               p3 -> subclass[A[intersection[RFX, image[S, singleton[x]]]],
             union[x, id[union[domain[x], range[x]]]]]}]]

Out[48]= or[not[member[x, V]], not[subclass[x, cart[V, V]]],
           subclass[A[intersection[RFX, image[S, singleton[x]]]],
             union[x, id[union[domain[x], range[x]]]]] == True

In[49]:= (% /. x -> x_) /. Equal -> SetDelayed
```

This can be sharpened to an equation:

```
In[50]:= SubstTest[and, implies[p, subclass[u, v]], subclass[v, u],
           {p -> member[x, P[cart[V, V]]], u -> A[intersection[RFX, image[S, singleton[x]]]],
          v -> union[x, id[union[domain[x], range[x]]]]}] // Reverse

Out[50]= or[equal[A[intersection[RFX, image[S, singleton[x]]]],
           union[x, id[union[domain[x], range[x]]]]],
         not[member[x, V]], not[subclass[x, cart[V, V]]]] == True

In[51]:= (% /. x -> x_) /. Equal -> SetDelayed

In[52]:= equal[intersection[complement[image[V, intersection[x, complement[cart[V, V]]]]],
           singleton[A[intersection[RFX, image[S, singleton[x]]]]]],
          singleton[A[intersection[RFX, image[S, singleton[x]]]]]]

Out[52]= True

In[53]:= intersection[complement[image[V, intersection[x_, complement[cart[V, V]]]]],
           singleton[A[intersection[RFX, image[S, singleton[x_]]]]]] :=
          singleton[A[intersection[RFX, image[S, singleton[x]]]]]
```

Note that:

```
In[54]:= equal[union[A[intersection[RFX, image[S, singleton[x]]]],
           complement[intersection[complement[
              image[V, intersection[x, complement[cart[V, V]]]]], image[V, singleton[x]]]]],
         union[x, id[union[domain[x], range[x]]], complement[intersection[complement[
              image[V, intersection[x, complement[cart[V, V]]]]], image[V, singleton[x]]]]]]

Out[54]= True
```

Replace **equal** with **Equal**, and map with **singleton**:

```
In[55]:= Map[singleton, Equal[union[
            A[intersection[RFX, image[S, singleton[x]]]], complement[intersection[complement[
               image[V, intersection[x, complement[cart[V, V]]]]], image[V, singleton[x]]]]],
         union[x, id[union[domain[x], range[x]]], complement[intersection[complement[
              image[V, intersection[x, complement[cart[V, V]]]]], image[V, singleton[x]]]]]]]

Out[55]= singleton[A[intersection[RFX, image[S, singleton[x]]]]] ==
          intersection[complement[image[V, intersection[x, complement[cart[V, V]]]]],
           singleton[union[x, id[union[domain[x], range[x]]]]]]

In[56]:= singleton[A[intersection[RFX, image[S, singleton[x_]]]]] :=
          intersection[complement[image[V, intersection[x, complement[cart[V, V]]]]],
           singleton[union[x, id[union[domain[x], range[x]]]]]]

In[57]:= dif[HULL[RFX], composite[CUP, id[composite[CUP, cross[IMAGE[DUP], IMAGE[DUP]], DORA]],
           inverse[FIRST], id[P[cart[V, V]]]]] // VSNormality

Out[57]= intersection[composite[complement[CUP],
           id[composite[CUP, cross[IMAGE[DUP], IMAGE[DUP]], DORA, id[P[cart[V, V]]]]],
           inverse[FIRST]], HULL[RFX]] == 0

In[58]:= intersection[composite[complement[CUP],
           id[composite[CUP, cross[IMAGE[DUP], IMAGE[DUP]], DORA, id[P[cart[V, V]]]]],
           inverse[FIRST]], HULL[RFX]] := 0
```

```
In[59]:= SubstTest[equal, 0, dif[u, v], {u -> HULL[RFX],
           v -> composite[CUP, id[composite[CUP, cross[IMAGE[DUP], IMAGE[DUP]], DORA]],
             inverse[FIRST], id[P[cart[V, V]]]]}] // Reverse

Out[59]= subclass[HULL[RFX], composite[CUP,
           id[composite[CUP, cross[IMAGE[DUP], IMAGE[DUP]], DORA, id[P[cart[V, V]]]]],
           inverse[FIRST]]] == True

In[60]:= % /. Equal -> SetDelayed
```

Inclusions of functions can be sharpened to equations:

```
In[61]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
          equal[u, composite[v, id[domain[u]]]],
            {u -> HULL[RFX],
           v -> composite[CUP, id[composite[CUP, cross[IMAGE[DUP], IMAGE[DUP]], DORA]],
             inverse[FIRST], id[P[cart[V, V]]]]}]

Out[61]= equal[composite[CUP,
           id[composite[CUP, cross[IMAGE[DUP], IMAGE[DUP]], DORA, id[P[cart[V, V]]]]],
           inverse[FIRST]], HULL[RFX]] == True
```

This yields a formula for the function **HULL[RFX]**.

```
In[62]:= composite[CUP,
           id[composite[CUP, cross[IMAGE[DUP], IMAGE[DUP]], DORA, id[P[cart[V, V]]]]],
           inverse[FIRST]] := HULL[RFX]
```

Corollary:

```
In[63]:= Assoc[IMAGE[id[Di]], CUP,
          composite[id[composite[CUP, cross[IMAGE[DUP], IMAGE[DUP]], DORA, id[P[cart[V, V]]]]],
           inverse[FIRST]]]

Out[63]= composite[IMAGE[id[Di]], HULL[RFX]] == composite[IMAGE[id[Di]], id[P[cart[V, V]]]]

In[64]:= composite[IMAGE[id[Di]], HULL[RFX]] := composite[IMAGE[id[Di]], id[P[cart[V, V]]]]
```

For completeness, the following additional fact about **HULL[RFX]** is derived, but it is not used further here.

```
In[65]:= SubstTest[implies, subclass[image[CUP, cart[x, x]], x],
             equal[composite[HULL[x], CUP], composite[CUP, cross[HULL[x], HULL[x]]]], x -> RFX]

Out[65]= equal[composite[CUP, cross[HULL[RFX], HULL[RFX]]], composite[HULL[RFX], CUP]] == True

In[66]:= composite[HULL[RFX], CUP] := composite[CUP, cross[HULL[RFX], HULL[RFX]]]
```

digression

The last result of the preceding section can be generalized.

```
In[67]:= SubstTest[implies, subclass[image[CUP, cart[y, y]], y], equal[
           composite[HULL[y], CUP], composite[CUP, cross[HULL[y], HULL[y]]]], y -> Uclosure[x]]

Out[67]= equal[composite[CUP, cross[HULL[Uclosure[x]], HULL[Uclosure[x]]]],
           composite[HULL[Uclosure[x]], CUP]] == True

In[68]:= composite[HULL[Uclosure[x_]], CUP] :=
          composite[CUP, cross[HULL[Uclosure[x]], HULL[Uclosure[x]]]]
```

Some applications:

```
In[69]:= SubstTest[composite, HULL[Uclosure[x]], CUP, x -> SYM]

Out[69]= composite[HULL[SYM], CUP] == composite[CUP, cross[HULL[SYM], HULL[SYM]]]

In[70]:= composite[HULL[SYM], CUP] := composite[CUP, cross[HULL[SYM], HULL[SYM]]]

In[71]:= SubstTest[composite, HULL[Uclosure[y]], CUP, y -> P[x]]

Out[71]= composite[id[P[x]], CUP] == composite[CUP, id[cart[P[x], P[x]]]]

In[72]:= composite[id[P[x_]], CUP] := composite[CUP, id[cart[P[x], P[x]]]]

In[73]:= SubstTest[composite, HULL[Uclosure[y]], CUP, y -> invar[x]]

Out[73]= composite[HULL[invar[x]], CUP] == composite[CUP, cross[HULL[invar[x]], HULL[invar[x]]]]

In[74]:= composite[HULL[invar[x_]], CUP] :=
          composite[CUP, cross[HULL[invar[x]], HULL[invar[x]]]]

In[75]:= SubstTest[composite, HULL[Uclosure[y]], CUP, y -> subvar[x]]

Out[75]= composite[HULL[subvar[x]], CUP] ==
          composite[CUP, cross[HULL[subvar[x]], HULL[subvar[x]]]]

In[76]:= composite[HULL[subvar[x_]], CUP] :=
          composite[CUP, cross[HULL[subvar[x]], HULL[subvar[x]]]]
```

The same technique also works in the opposite direction:

```
In[77]:= ImageComp[HULL[Uclosure[x]], CUP, V]

Out[77]= image[CUP, cart[fix[HULL[Uclosure[x]]], fix[HULL[Uclosure[x]]]]] ==
          fix[HULL[Uclosure[x]]]

In[78]:= image[CUP, cart[fix[HULL[Uclosure[x_]]], fix[HULL[Uclosure[x_]]]]] :=
          fix[HULL[Uclosure[x]]]
```

irreflexive transitive relations

A partial order is a reflexive, antisymmetric, transitive relation. Corresponding to any partial order **x**, there is an irreflexive, transitive relation **intersection[Di,x]**, and conversely, for each irreflexive, transitive relation **x**, there is a partial ordering **union[x,id[union[domain[x],range[x]]]]**. This is not a one−to−one correspondence because one can always add any subclass of **Id** to a partial order to obtain another partial order corresponding to the same irreflexive transitive relation.

```
In[79]:= implies[PARTIALORDER[x], PARTIALORDER[union[x, id[y]]]] // not // not

Out[79]= True

In[80]:= ((implies[and[member[x, PO], member[y, V]], member[union[x, id[y]], PO]] // not //
          NotNotTest) /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

```
In[81]:= image[inverse[ADJOIN[id[y]]], z] // Normality // Reverse

Out[81]= fix[composite[S, IMAGE[id[union[Di, complement[cart[y, V]]]]],
            id[intersection[z, image[inverse[IMAGE[inverse[DUP]]], image[S, singleton[y]]]]],
            S]] == image[inverse[ADJOIN[id[y]]], z]

In[82]:= fix[composite[S, IMAGE[id[union[Di, complement[cart[y_, V]]]]],
            id[intersection[z_, image[inverse[IMAGE[inverse[DUP]]], image[S, singleton[y_]]]]],
            S]] := image[inverse[ADJOIN[id[y]]], z]
```

Eliminating the variables is a bit tricky. Here is one way to do it:

```
In[83]:= Map[equal[0, #] &, SubstTest[class, x,
            implies[and[member[x, z], member[y, V]], member[union[x, id[y]], z]] // not,
            z -> PO]] // Reverse

Out[83]= or[not[member[y, V]], subclass[PO, image[inverse[ADJOIN[id[y]]], PO]]] == True

In[84]:= or[not[member[y_, V]], subclass[PO, image[inverse[ADJOIN[id[y_]]], PO]]] := True

In[85]:= class[x, subclass[z, image[inverse[ADJOIN[x]], z]]]

Out[85]= complement[image[complement[image[inverse[CUP], z]], z]]

In[86]:= implies[member[y, V],
           member[id[y], complement[image[complement[image[inverse[CUP], PO]], PO]]]]

Out[86]= True

In[87]:= Map[equal[V, #] &, SubstTest[class, y, implies[member[y, V], member[id[y], z]],
            z -> complement[image[complement[image[inverse[CUP], PO]], PO]]] // Reverse

Out[87]= subclass[image[CUP, cart[PO, P[Id]]], PO] == True

In[88]:= subclass[image[CUP, cart[PO, P[Id]]], PO] := True
```

This result can be simplified further:

```
In[89]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
             {u -> cart[PO, singleton[0]], v -> cart[PO, P[Id]], w -> CUP}]

Out[89]= subclass[PO, image[CUP, cart[PO, P[Id]]]] == True

In[90]:= subclass[PO, image[CUP, cart[PO, P[Id]]]] := True

In[91]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u -> image[CUP, cart[PO, P[Id]]], v -> PO}]

Out[91]= True == equal[PO, image[CUP, cart[PO, P[Id]]]]

In[92]:= image[CUP, cart[PO, P[Id]]] := PO
```

definitions

A relation x is **irreflexive** if

```
In[93]:= assert[forall[y, not[member[pair[y, y], x]]]]

Out[93]= equal[0, fix[x]]
```

The class of irreflexive relations is

```
In[94]:= class[x, and[subclass[x, cart[V, V]], equal[0, fix[x]]]]

Out[94]= P[Di]
```

The **GOEDEL** program already contains this fact:

```
In[95]:= image[IMAGE[id[Di]], intersection[ANTISYM, TRV]]

Out[95]= intersection[TRV, P[Di]]
```

Our immediate aim is to show that **image[IMAGE[id[Di]], PO] = intersection[TRV, P[Di]]**. This will be done by showing that each side of the equation is contained in the other. One direction is easy:

```
In[96]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
             {u -> PO, v -> intersection[ANTISYM, TRV], w -> IMAGE[id[Di]]}]

Out[96]= subclass[image[IMAGE[id[Di]], PO], TRV] == True

In[97]:= subclass[image[IMAGE[id[Di]], PO], TRV] := True
```

This completes the proof of inclusion in one direction.

```
In[98]:= subclass[image[IMAGE[id[Di]], PO], intersection[TRV, P[Di]]]

Out[98]= True
```

inclusion in the other direction

```
In[99]:= Map[not, SubstTest[and, implies[p1, p3],
            implies[and[p2, p3], p4], not[implies[and[p1, p2], p4]],
              {p1 -> subclass[composite[x, x], x], p2 -> equal[0, fix[x]],
             p3 -> subclass[fix[composite[x, x]], fix[x]],
                  p4 -> equal[0, fix[composite[x, x]]]}]]

Out[99]= or[equal[0, fix[composite[x, x]]],
            not[equal[0, fix[x]]], not[subclass[composite[x, x], x]]] == True

In[100]:=
        (% /. x -> x_) /. Equal -> SetDelayed

In[101]:=
        or[not[equal[0, fix[x]]],
          not[subclass[composite[x, x], x]], disjoint[x, inverse[x]]] // AssertTest

Out[101]=
        or[equal[0, intersection[x, inverse[x]]],
          not[equal[0, fix[x]]], not[subclass[composite[x, x], x]]] == True

In[102]:=
        (% /. x -> x_) /. Equal -> SetDelayed
```

```
In[103]:=
Map[not, SubstTest[and, implies[and[p1, p2], p3],
  implies[p3, p4], not[implies[and[p1, p2], p4]],
    {p1 -> subclass[composite[x, x], x], p2 -> equal[0, fix[x]],
   p3 -> disjoint[x, inverse[x]],
        p4 -> subclass[intersection[x, inverse[x]], Id]}]]

Out[103]=
or[not[equal[0, fix[x]]], not[subclass[composite[x, x], x]],
  subclass[intersection[x, inverse[x]], Id]] == True

In[104]:=
(% /. x -> x_) /. Equal -> SetDelayed

In[105]:=
Map[implies[member[x, intersection[TRV, P[Di]]], #] &,
  SubstTest[member, union[x, id[union[domain[x], range[x]]]],
            intersection[u, v, w], {u -> RFX, v -> ANTISYM, w -> TRV}]] // MapNotNot

Out[105]=
or[not[equal[0, fix[x]]], not[member[x, V]],
  not[subclass[x, cart[V, V]]], not[subclass[composite[x, x], x]],
  TRANSITIVE[union[x, id[union[domain[x], range[x]]]]]] == True

In[106]:=
(% /. x -> x_) /. Equal -> SetDelayed

In[107]:=
implies[member[x, intersection[TRV, P[Di]]],
  member[union[x, id[union[domain[x], range[x]]]], PO]] // NotNotTest

Out[107]=
or[and[subclass[x, cart[union[domain[x], range[x]], union[domain[x], range[x]]]],
   subclass[intersection[x, inverse[x]], Id],
   TRANSITIVE[union[x, id[union[domain[x], range[x]]]]]],
  not[equal[0, fix[x]]], not[member[x, V]], not[subclass[x, cart[V, V]]],
  not[subclass[composite[x, x], x]]] == True

In[108]:=
(% /. x -> x_) /. Equal -> SetDelayed
```

This takes a while:

```
In[109]:=
implies[and[member[w, RFX], member[w, TRV], member[w, ANTISYM]], member[w, PO]] //
 AssertTest

Out[109]=
or[not[member[w, V]],
  not[subclass[w, cart[fix[w], fix[w]]]], not[subclass[composite[w, w], w]],
  not[subclass[intersection[w, inverse[w]], Id]], TRANSITIVE[w]] == True

In[110]:=
(% /. w -> w_) /. Equal -> SetDelayed

In[111]:=
SubstTest[implies,
 and[member[w, RFX], member[w, TRV], member[w, ANTISYM]], member[w, PO],
    {w -> union[x, id[union[domain[x], range[x]]]]}]

Out[111]=
or[not[member[x, V]],
  not[subclass[x, cart[union[domain[x], range[x]], union[domain[x], range[x]]]]],
  not[subclass[composite[x, x], union[x, id[union[domain[x], range[x]]]]]],
  not[subclass[intersection[x, inverse[x]], Id]],
  TRANSITIVE[union[x, id[union[domain[x], range[x]]]]]] == True
```

In[112]:=
```
(% /. x -> x_) /. Equal -> SetDelayed
```

In[113]:=
```
equiv[or[not[member[x, V]], not[member[intersection[Di, x], V]]], or[not[member[x, V]]]]
```

Out[113]=
```
True
```

In[114]:=
```
or[not[member[x_, V]], not[member[intersection[Di, x_], V]]] := not[member[x, V]]
```

In[115]:=
```
SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
    {u -> singleton[union[x, id[union[domain[x], range[x]]]]],
  v -> PO, w -> IMAGE[id[Di]]}]
```

Out[115]=
```
or[member[intersection[Di, x], image[IMAGE[id[Di]], PO]], not[member[x, V]],
  not[subclass[x, cart[union[domain[x], range[x]], union[domain[x], range[x]]]]],
  not[subclass[intersection[x, inverse[x]], Id]],
  not[TRANSITIVE[union[x, id[union[domain[x], range[x]]]]]] == True
```

In[116]:=
```
(% /. x -> x_) /. Equal -> SetDelayed
```

In[117]:=
```
SubstTest[implies, and[equal[x, y], member[y, z]],
 member[x, z], {y -> intersection[Di, x], z -> image[IMAGE[id[Di]], PO]}]
```

Out[117]=
```
or[member[x, image[IMAGE[id[Di]], PO]], not[equal[0, fix[x]]],
  not[member[intersection[Di, x], image[IMAGE[id[Di]], PO]]],
  not[subclass[x, cart[V, V]]] == True
```

In[118]:=
```
(% /. x -> x_) /. Equal -> SetDelayed
```

In[119]:=
```
Map[not, SubstTest[and, implies[p1, p2],
  implies[p2, p3], implies[and[p1, p3], p4], not[implies[p1, p4]],
    {p1 -> member[x, intersection[TRV, P[Di]]],
   p2 -> member[union[x, id[union[domain[x], range[x]]]], PO],
        p3 -> member[intersection[Di, x], image[IMAGE[id[Di]], PO]],
   p4 -> member[x, image[IMAGE[id[Di]], PO]]}]]
```

Out[119]=
```
or[member[x, image[IMAGE[id[Di]], PO]], not[equal[0, fix[x]]], not[member[x, V]],
  not[subclass[x, cart[V, V]]], not[subclass[composite[x, x], x]]] == True
```

In[120]:=
```
(% /. x -> x_) /. Equal -> SetDelayed
```

In[121]:=
```
Map[equal[V, #] &, SubstTest[class, x, implies[member[x, y], member[x, z]],
   {y -> intersection[TRV, P[Di]], z -> image[IMAGE[id[Di]], PO]}]] // Reverse
```

Out[121]=
```
subclass[intersection[TRV, P[Di]], image[IMAGE[id[Di]], PO]] == True
```

In[122]:=
```
subclass[intersection[TRV, P[Di]], image[IMAGE[id[Di]], PO]] := True
```

```
In[123]:=
SubstTest[and, subclass[u, v], subclass[v, u],
    {u -> intersection[TRV, P[Di]], v -> image[IMAGE[id[Di]], PO]}] // Reverse
```

```
Out[123]=
equal[image[IMAGE[id[Di]], PO], intersection[TRV, P[Di]]] == True
```

This completes the derivation.

```
In[124]:=
image[IMAGE[id[Di]], PO] := intersection[TRV, P[Di]]
```

This equation says that every irreflexive transitive relation can be obtained by removing an identity relation from a partial ordering.

other direction

In the other direction, from every irreflexive partial ordering **x** one can construct a partial ordering. This partial order is not unique, but there is a least such, namely the reflexive hull of **x**.

```
In[125]:=
subclass[image[HULL[RFX], intersection[TRV, P[Di]]], PO]
```

```
Out[125]=
subclass[image[HULL[RFX], intersection[TRV, P[Di]]], PO]
```

```
In[126]:=
(SubstTest[implies, and[equal[u, v], member[v, w]], member[u, w],
    {u -> hull[RFX, x], v -> union[x, id[union[domain[x], range[x]]]], w -> PO}] /.
   x -> x_) /. Equal -> SetDelayed
```

```
In[127]:=
Map[not, SubstTest[and, implies[p1, p2],
  implies[p2, p3], implies[p1, p4], implies[and[p3, p4], p5],
       not[implies[p1, p5]],
    {p1 -> member[x, intersection[TRV, P[Di]]], p2 -> member[x, P[cart[V, V]]],
       p3 -> equal[hull[RFX, x], union[x, id[union[domain[x], range[x]]]]],
       p4 -> member[union[x, id[union[domain[x], range[x]]]], PO],
       p5 -> member[hull[RFX, x], PO]}]]
```

```
Out[127]=
or[and[subclass[A[intersection[RFX, image[S, singleton[x]]]],
    cart[fix[A[intersection[RFX, image[S, singleton[x]]]]],
     fix[A[intersection[RFX, image[S, singleton[x]]]]]]],
   subclass[intersection[A[intersection[RFX, image[S, singleton[x]]]],
     inverse[A[intersection[RFX, image[S, singleton[x]]]]], Id],
    TRANSITIVE[A[intersection[RFX, image[S, singleton[x]]]]],
  not[equal[0, fix[x]]], not[member[x, V]], not[subclass[x, cart[V, V]]],
  not[subclass[composite[x, x], x]]] == True
```

```
In[128]:=
(% /. x -> x_) /. Equal -> SetDelayed
```

```
In[129]:=
SubstTest[class, x, implies[member[x, y], member[hull[RFX, x], z]],
 {y -> intersection[TRV, P[Di]], z -> PO}]
```

```
Out[129]=
V == union[complement[TRV], complement[P[Di]], image[inverse[HULL[RFX]], PO]]
```

```
In[130]:=
Map[equal[V, #] &, %] // Reverse

Out[130]=
subclass[intersection[TRV, P[Di]], image[inverse[HULL[RFX]], PO]] == True

In[131]:=
subclass[intersection[TRV, P[Di]], image[inverse[HULL[RFX]], PO]] := True

In[132]:=
SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
    {u -> intersection[TRV, P[Di]], v -> image[inverse[HULL[RFX]], PO], w -> HULL[RFX]}]

Out[132]=
and[subclass[image[HULL[RFX], intersection[TRV, P[Di]]], PO],
  subclass[image[HULL[RFX], intersection[TRV, P[Di]]], RFX]] == True

In[133]:=
subclass[image[HULL[RFX], intersection[TRV, P[Di]]], PO] := True
```

This inclusion cannot be strengthened to an equation because, for instance, the left side does not contain any identities, but the right side does.