similar relations

Johan G. F. Belinfante
2010 February 26

```
In[1]:= SetDirectory["l:"]; << goedel.10feb25a; << tools.m

        :Package Title: goedel.10feb25a                2010 February 25 at 6:20 p.m.

        It is now:  2010 Feb 26 at 7:53

        Loading Simplification Rules

        TOOLS.M                                    Revised 2010 January 29

        weightlimit = 40
```

summary

A challenging obstacle to developing a theory of similar relations in the **GOEDEL** program was to formulate a definition of similarity that resembles definitions commonly found in the literature and at the same time does not lead to excessive execution times for basic results such as normalization. It is common in the mathematical literature to consider not just relations, but ordered pairs consisting of a relation and a set whose cartesian square contains the given relation. Extra variables are also introduced to permit the use of function application in defining similarity. See for example Definition 1 on page 128 in the following book.

```
In[2]:= "Patrick Suppes, Axiomatic Set Theory, Dover Publications, New York, 1972.";
```

A key idea for reducing execution time is to define similarity without introducing unneeded variables. A relation **y** is **similar** to a relation **x** if there exists a one-to-one function **s** such that **y = s ∘ x ∘ inverse[s]** and the domain and range of **x** are both contained in the domain of **s**. In this case **t = s ⊗ s** is also a one-to-one function and satisfies the simpler conditions **y = image[t, x]** and **x ⊂ domain[t]**. The class of all such one-to-one functions **t** is **image[CROSS, id[BIJ]]**. These considerations motivated defining the similarity relation **SIMILAR** by the following **class**-wrapped membership rule that has been added to the **GOEDEL** program. Sequestering **image[CROSS, id[BIJ]]** helps to speed up the derivation of the normalization rule.

```
In[3]:= Begin["Goedel`Private`"];

In[4]:= FirstMatch[class[r_, member[x_, HoldPattern[SIMILAR]]]]

Out[4]= class[r_, member[x_, SIMILAR]] := Module[{s = Unique[], t = Unique[]},
          class[r, exists[s, and[equal[second[x], image[s, first[x]]], member[
          s, t], subclass[first[x], domain[s]]]]] /. t -> image[CROSS, id[BIJ]]]
```

In this notebook it is shown that **SIMILAR** is an equivalence relation and that similar relations are equipollent. It is shown that if two sets are equipollent, then their cartesian squares are similar, that any partial order is similar to a restriction of the subset relation **S** and that any relation similar to a well-order is a well-order. Analogous results also hold for other classes

of relations.

On a technical note it should be pointed out that many basic properties of similarity are derived in this notebook by making good use of the function **IPD = λ x. IMAGE[x] ∘ id[P[domain[x]]]** that had originally been introduced to define the power functor in the category of sets. It was recently shown that the equipollence relation **Q** is related to **IPD** as follows:

```
In[5]:= U[image[IPD, BIJ]]
```

```
Out[5]= Q
```

It is shown below that the similarity relation is obtained by simply simply replacing the class **BIJ** of all bijections in this formula by the subclass **image[CROSS, id[BIJ]] ⊂ BIJ**. This result is without doubt the most important consequence of the normalization rule.

normalization

Clearing the **simplify** and **cond** flags reduces the execution time for the normalization rule from 21 seconds to about 8 seconds.

```
In[6]:= simplify = False; cond = False;
```

Theorem. Normalization rule.

```
In[7]:= SIMILAR // Normality // Reverse
```

```
Out[7]= composite[IMG, id[composite[inverse[S], IMAGE[FIRST], id[image[CROSS, id[BIJ]]]]],
          inverse[SECOND]] == SIMILAR
```

```
In[8]:= composite[IMG, id[composite[inverse[S], IMAGE[FIRST], id[image[CROSS, id[BIJ]]]]],
          inverse[SECOND]] := SIMILAR
```

The cleared flags are now reset.

```
In[9]:= simplify = True; cond = True;
```

Observation. The following compact restatement of the normalization rule for **SIMILAR** is often useful.

```
In[10]:= U[image[IPD, image[CROSS, id[BIJ]]]]
```

```
Out[10]= SIMILAR
```

basic properties

Theorem. The class **SIMILAR** is a relation.

```
In[11]:= SubstTest[subclass, U[image[IPD, x]], cart[V, V], x -> image[CROSS, id[BIJ]]] // Reverse
Out[11]= subclass[SIMILAR, cart[V, V]] == True

In[12]:= % /. Equal→ SetDelayed
```

Corollary. A simplification rule.

```
In[13]:= SubstTest[composite, Id, U[image[IPD, x]], x -> image[CROSS, id[BIJ]]] // Reverse
Out[13]= composite[Id, SIMILAR] == SIMILAR

In[14]:= composite[Id, SIMILAR] := SIMILAR
```

Theorem. The only relation similar to the empty relation is the empty relation itself.

```
In[15]:= image[SIMILAR, set[0]] // Normality
Out[15]= image[SIMILAR, set[0]] == set[0]

In[16]:= image[SIMILAR, set[0]] := set[0]
```

Corollary. The empty relation is similar to itself.

```
In[17]:= member[pair[0, 0], SIMILAR] // AssertTest
Out[17]= member[pair[0, 0], SIMILAR] == True

In[18]:= member[pair[0, 0], SIMILAR] := True
```

Lemma. Simplification rule.

```
In[19]:= ImageComp[IMAGE[FIRST], CROSS, id[BIJ]] // Reverse
Out[19]= image[IMAGE[FIRST], image[CROSS, id[BIJ]]] == image[CART, Id]

In[20]:= image[IMAGE[FIRST], image[CROSS, id[BIJ]]] := image[CART, Id]
```

Theorem. The domain of **SIMILAR** is the class of all (small) relations.

```
In[21]:= SubstTest[domain, U[image[IPD, x]], x -> image[CROSS, id[BIJ]]] // Reverse
Out[21]= domain[SIMILAR] == P[cart[V, V]]

In[22]:= domain[SIMILAR] := P[cart[V, V]]
```

closure under composition

The composite of bijections is a bijection. Some variable-free consequences of this and related results are derived in this section.

Lemma.

```
In[23]:= Assoc[COMPOSE, cross[CROSS, CROSS], cross[DUP, DUP]]

Out[23]= composite[COMPOSE, cross[composite[CROSS, DUP], composite[CROSS, DUP]]] ==
          composite[CROSS, DUP, COMPOSE]

In[24]:= composite[COMPOSE, cross[composite[CROSS, DUP], composite[CROSS, DUP]]] :=
          composite[CROSS, DUP, COMPOSE]
```

Theorem. The class **image[CROSS, id[BIJ]]** is closed under composition.

```
In[25]:= ImageComp[COMPOSE,
           cross[composite[CROSS, DUP], composite[CROSS, DUP]], cart[BIJ, BIJ]] // Reverse

Out[25]= image[COMPOSE, cart[image[CROSS, id[BIJ]], image[CROSS, id[BIJ]]]] ==
          image[CROSS, id[BIJ]]

In[26]:= image[COMPOSE, cart[image[CROSS, id[BIJ]], image[CROSS, id[BIJ]]]] :=
          image[CROSS, id[BIJ]]
```

Although the constructor **f[x] = IMAGE[x] ∘ id[P[domain[x]]]** does not preserve composition in general, it does so when the right-hand factor is a function. In particular, this constructor preserves composition of bijections. The following variable-free corollary follows from a similar rewrite rule derived in the course of studying the power functor in the category of sets.

Theorem. the constructor **f[x] = IMAGE[x] ∘ id[P[domain[x]]]** preserves composition of bijections.

```
In[27]:= Assoc[composite[IPD, COMPOSE], cross[Id, FUNPART], cross[OOPART, OOPART]]

Out[27]= composite[IPD, COMPOSE, cross[OOPART, OOPART]] ==
          composite[COMPOSE, cross[composite[IPD, OOPART], composite[IPD, OOPART]]]

In[28]:= composite[IPD, COMPOSE, cross[OOPART, OOPART]] :=
          composite[COMPOSE, cross[composite[IPD, OOPART], composite[IPD, OOPART]]]
```

Corollary. The class **image[IPD, BIJ]** is binary-closed under **COMPOSE**.

```
In[29]:= ImageComp[composite[IPD, COMPOSE], cross[OOPART, OOPART], V]

Out[29]= image[COMPOSE, cart[image[IPD, BIJ], image[IPD, BIJ]]] == image[IPD, BIJ]

In[30]:= image[COMPOSE, cart[image[IPD, BIJ], image[IPD, BIJ]]] := image[IPD, BIJ]
```

Theorem. The subclass **image[CROSS, id[BIJ]] ⊂ BIJ** is binary-closed under composition.

```
In[31]:= Map[range, Assoc[composite[IPD, COMPOSE], cross[OOPART, OOPART],
            cross[composite[CROSS, DUP, OOPART], composite[CROSS, DUP, OOPART]]]] // Reverse

Out[31]= image[COMPOSE,
           cart[image[IPD, image[CROSS, id[BIJ]]], image[IPD, image[CROSS, id[BIJ]]]]] ==
          image[IPD, image[CROSS, id[BIJ]]]

In[32]:= image[COMPOSE, cart[image[IPD, image[CROSS, id[BIJ]]],
            image[IPD, image[CROSS, id[BIJ]]]]] := image[IPD, image[CROSS, id[BIJ]]]
```

similarity is an equivalence relation

The final rewrite rule of the preceding section implies that **SIMILAR** is idempotent.

Theorem. The similarity relation is idempotent.

```
In[33]:= SubstTest[U, image[COMPOSE, cart[x, x]], x -> image[IPD, image[CROSS, id[BIJ]]]]

Out[33]= composite[SIMILAR, SIMILAR] == SIMILAR

In[34]:= composite[SIMILAR, SIMILAR] := SIMILAR
```

Corollary.

```
In[35]:= SubstTest[subclass, composite[x, x], x, x → SIMILAR]

Out[35]= TRANSITIVE[SIMILAR] == True

In[36]:= TRANSITIVE[SIMILAR] := True
```

In the following corollary three varuables are introduce to restate the transitivity of **SIMILAR**.

Corollary.

```
In[37]:= SubstTest[implies, and[member[pair[x, y], trv[t]], member[pair[y, z], trv[t]]],
           member[pair[x, z], trv[t]], t → SIMILAR] // Reverse

Out[37]= or[member[pair[x, z], SIMILAR],
           not[member[pair[x, y], SIMILAR]], not[member[pair[y, z], SIMILAR]]] == True

In[38]:= or[member[pair[x_, z_], SIMILAR],
           not[member[pair[x_, y_], SIMILAR]], not[member[pair[y_, z_], SIMILAR]]] := True
```

Theorem. The relation **SIMILAR** is symmetric.

```
In[39]:= Map[U[image[#, V]] &,
           Assoc[composite[IPD, IMAGE[SWAP]], OOPART, composite[CROSS, DUP, OOPART]]] // Reverse

Out[39]= inverse[SIMILAR] == SIMILAR

In[40]:= inverse[SIMILAR] := SIMILAR
```

Corollary. Restatement of symmetry using variables.

```
In[41]:= Map[implies[member[pair[y, x], SIMILAR], #] &,
           SubstTest[member, pair[x, y], inverse[t], t → SIMILAR]] // Reverse

Out[41]= or[member[pair[x, y], SIMILAR], not[member[pair[y, x], SIMILAR]]] == True

In[42]:= or[member[pair[x_, y_], SIMILAR], not[member[pair[y_, x_], SIMILAR]]] := True
```

Corollary. The similarity relation is an equivalence relation.

```
In[43]:= SubstTest[and, SYMMETRIC[x], TRANSITIVE[x], x → SIMILAR]

Out[43]= EQUIVALENCE[SIMILAR] == True

In[44]:= EQUIVALENCE[SIMILAR] := True
```

Theorem. The range of **SIMILAR** is equal to its domain.

```
In[45]:= SubstTest[domain, inverse[t], t → SIMILAR]

Out[45]= range[SIMILAR] == P[cart[V, V]]

In[46]:= range[SIMILAR] := P[cart[V, V]]
```

Corollary.

```
In[47]:= SubstTest[subclass, composite[Id, t], cart[x, y], t → SIMILAR] // Reverse

Out[47]= subclass[SIMILAR, cart[x, y]] ==
          and[subclass[P[cart[V, V]], x], subclass[P[cart[V, V]], y]]

In[48]:= subclass[SIMILAR, cart[x_, y_]] :=
          and[subclass[P[cart[V, V]], x], subclass[P[cart[V, V]], y]]
```

Theorem. Every relation is similar to itself.

```
In[49]:= SubstTest[range, eqv[x], x → SIMILAR]

Out[49]= fix[SIMILAR] == P[cart[V, V]]

In[50]:= fix[SIMILAR] := P[cart[V, V]]
```

Corollary.

```
In[51]:= SubstTest[REFLEXIVE, eqv[x], x → SIMILAR] // Reverse

Out[51]= REFLEXIVE[SIMILAR] == True

In[52]:= REFLEXIVE[SIMILAR] := True
```

similarity implies equipollence

Theorem. Similar relations are equipollent.

```
In[53]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
           {t → composite[inverse[E], IPD], u -> image[CROSS, id[BIJ]], v → BIJ}] // Reverse

Out[53]= subclass[SIMILAR, Q] == True

In[54]:= subclass[SIMILAR, Q] := True
```

Lemma.

```
In[55]:= Map[image[#, id[BIJ]] &,
          Assoc[cross[IMAGE[FIRST], IMAGE[SECOND]], cross[CROSS, CROSS], DUP]]

Out[55]= image[DORA, image[CROSS, id[BIJ]]] ==
          composite[CART, DUP, Q, inverse[DUP], inverse[CART]]

In[56]:= image[DORA, image[CROSS, id[BIJ]]] :=
          composite[CART, DUP, Q, inverse[DUP], inverse[CART]]
```

Theorem. A lower bound.

```
In[57]:= SubstTest[implies, subclass[u, v], subclass[image[u, w], image[v, w]],
           {u → DORA, v → composite[inverse[E], IPD], w -> image[CROSS, id[BIJ]]}] // Reverse

Out[57]= subclass[composite[CART, DUP, Q, inverse[DUP], inverse[CART]], SIMILAR] == True

In[58]:= subclass[composite[CART, DUP, Q, inverse[DUP], inverse[CART]], SIMILAR] := True
```

Corollary.

```
In[59]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
           {t → inverse[cross[composite[CART, DUP], composite[CART, DUP]]],
            u -> composite[CART, DUP, Q, inverse[DUP], inverse[CART]], v -> SIMILAR}] // Reverse

Out[59]= subclass[Q, composite[inverse[DUP], inverse[CART], SIMILAR, CART, DUP]] == True

In[60]:= subclass[Q, composite[inverse[DUP], inverse[CART], SIMILAR, CART, DUP]] := True
```

The following lemma reintroduces variables. Using the **setpart** wrapper helps simplify the result.

Lemma.

```
In[61]:= SubstTest[implies, and[member[u, v], subclass[v, w]],
           member[u, w], {u → pair[setpart[x], setpart[y]], v → Q,
            w -> composite[inverse[DUP], inverse[CART], SIMILAR, CART, DUP]}] // Reverse

Out[61]= or[member[pair[cart[setpart[x], setpart[x]], cart[setpart[y], setpart[y]]], SIMILAR],
           not[member[pair[setpart[x], setpart[y]], Q]]] == True

In[62]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

A better result is obtained by removing the **setpart** wrappers.

Theorem. If two sets are equipollent, then their cartesian squares are similar.

```
In[63]:= SubstTest[implies, and[equal[x, setpart[u]], equal[y, setpart[v]]],
           or[member[pair[cart[x, x], cart[y, y]], SIMILAR], not[member[pair[x, y], Q]]],
           {u → x, v → y}] // Reverse

Out[63]= or[member[pair[cart[x, x], cart[y, y]], SIMILAR], not[member[pair[x, y], Q]]] == True
```

```
In[64]:= or[member[pair[cart[x_, x_], cart[y_, y_]], SIMILAR],
           not[member[pair[x_, y_], Q]]] := True
```

wellorders

In this section it is shown that similarity preserves the property of being a well-ordering. The same methods can be used to derive analogous results for the class of antisymmetric relations, the class of equivalence relations, the class of partial orders, the class of reflexive relations, the class of total orders, the class of transitive relations and others.

Theorem. An upper bound for **SIMILAR**.

```
In[65]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]], {t ->
             intersection[composite[inverse[FIRST], SECOND], composite[inverse[SECOND], IMG]],
            u → composite[inverse[S], IMAGE[FIRST], id[image[CROSS, id[BIJ]]]],
            v -> cart[image[CROSS, id[BIJ]], V]}] // Reverse

Out[65]= subclass[SIMILAR,
           composite[IMG, id[cart[image[CROSS, id[BIJ]], V]], inverse[SECOND]]] == True

In[66]:= subclass[SIMILAR,
           composite[IMG, id[cart[image[CROSS, id[BIJ]], V]], inverse[SECOND]]] := True
```

Lemma. An inclusion in one direction.

```
In[67]:= SubstTest[implies, subclass[u, v],
           subclass[image[u, w], image[v, w]], {u -> SIMILAR, v -> composite[IMG,
              id[cart[image[CROSS, id[BIJ]], V]], inverse[SECOND]], w → WO}] // Reverse

Out[67]= subclass[image[SIMILAR, WO], WO] == True

In[68]:= % /. Equal→ SetDelayed
```

The opposite inclusion is automatically recognized by the **GOEDEL** program via conditional rewrite rules, and therefore an equation can be derived and made into a rewrite rule.

Theorem. Any relation similar to a well-order is a well-order.

```
In[69]:= SubstTest[and, subclass[u, v], subclass[v, u], {u -> image[SIMILAR, WO], v → WO}]

Out[69]= equal[WO, image[SIMILAR, WO]] == True

In[70]:= image[SIMILAR, WO] := WO
```

reintroducing variables

In this section variables are reintroduced to provide a rewrite rule for membership in the similarity relation.

Lemma. This lemma introduces a variable **x** wrapped with **oopart** and **setpart**.

```
In[71]:= SubstTest[implies, subclass[u, v],
           subclass[image[t, u], image[t, v]], {t -> composite[inverse[E], IPD, CROSS, DUP],
            u → set[oopart[setpart[x]]], v → BIJ}] // Reverse

Out[71]= subclass[composite[IMAGE[cross[oopart[setpart[x]], oopart[setpart[x]]]], id[
           P[cart[domain[oopart[setpart[x]]], domain[oopart[setpart[x]]]]]], SIMILAR] == True

In[72]:= (% /. x → x_) /. Equal→ SetDelayed
```

Theorem. (Eliminating wrappers.)

```
In[73]:= Map[implies[member[x, y], #] &,
           SubstTest[implies, equal[x, oopart[setpart[t]]], subclass[composite[
              IMAGE[cross[x, x]], id[P[cart[domain[x], domain[x]]]], SIMILAR], t → x]] // Reverse

Out[73]= or[not[FUNCTION[x]], not[FUNCTION[inverse[x]]], not[member[x, y]], subclass[
            composite[IMAGE[cross[x, x]], id[P[cart[domain[x], domain[x]]]], SIMILAR]] == True

In[74]:= or[not[FUNCTION[x_]], not[FUNCTION[inverse[x_]]],
           not[member[x_, y_]], subclass[composite[IMAGE[cross[x_, x_]],
             id[P[cart[domain[x_], domain[x_]]]], SIMILAR]] := True
```

Lemma. This lemma introduces a second variable **u** which is a relation.

```
In[75]:= (SubstTest[implies, and[member[w, y], subclass[y, z]], member[w, z], {w → pair[u, v],
              y -> composite[IMAGE[cross[oopart[setpart[x]], oopart[setpart[x]]]],
                id[P[cart[domain[oopart[setpart[x]]], domain[oopart[setpart[x]]]]]]],
              z -> SIMILAR}] // Reverse) /. v -> composite[
            oopart[setpart[x]], u, inverse[oopart[setpart[x]]]]

Out[75]= or[member[pair[u, composite[oopart[setpart[x]], u, inverse[oopart[setpart[x]]]]],
            SIMILAR], not[member[u, V]],
           not[member[composite[oopart[setpart[x]], u, inverse[oopart[setpart[x]]]], V]], not[
            subclass[u, cart[domain[oopart[setpart[x]]], domain[oopart[setpart[x]]]]]] == True

In[76]:= (% /. {u → u_, x → x_}) /. Equal→ SetDelayed
```

Lemma. Eliminating wrappers.

```
In[77]:= SubstTest[implies, equal[x, oopart[setpart[t]]],
           or[member[pair[u, composite[x, u, inverse[x]]], SIMILAR],
            not[member[u, V]], not[member[composite[x, u, inverse[x]], V]],
            not[subclass[u, cartsq[domain[x]]]]], t → x] // Reverse

Out[77]= or[member[pair[u, composite[x, u, inverse[x]]], SIMILAR],
           not[FUNCTION[x]], not[FUNCTION[inverse[x]]], not[member[u, V]],
           not[member[x, V]], not[member[composite[x, u, inverse[x]], V]],
           not[subclass[u, cart[domain[x], domain[x]]]] == True

In[78]:= (% /. {u → u_, x → x_}) /. Equal→ SetDelayed
```

Lemma. Eliminating an unneeded sethood literal.

```
In[79]:= Map[implies[and[member[x, u], member[y, v]], #] &, Map[not, SubstTest[and,
           implies[p1, p4], implies[and[p1, p2, p3, p4], p5], not[implies[and[p1, p2, p3], p5]],
           {p1 → and[FUNCTION[y], FUNCTION[inverse[y]], member[y, V]],
            p2 → member[x, V], p3 → subclass[x, cart[domain[y], domain[y]]],
            p4 → member[composite[y, x, inverse[y]], V],
            p5 → member[pair[x, composite[y, x, inverse[y]]], SIMILAR]}]]] // Reverse

Out[79]= or[member[pair[x, composite[y, x, inverse[y]]], SIMILAR],
          not[FUNCTION[y]], not[FUNCTION[inverse[y]]], not[member[x, u]],
          not[member[y, v]], not[subclass[x, cart[domain[y], domain[y]]]]] == True

In[80]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. Any subclass of the cartesian square of a set is a set.

```
In[81]:= Map[implies[member[y, z], #] &, SubstTest[implies,
           and[subclass[x, t], member[t, V]], member[x, V], t → cartsq[y]]] // Reverse

Out[81]= or[member[x, V], not[member[y, z]], not[subclass[x, cart[y, y]]]] == True

In[82]:= or[member[x_, V], not[member[y_, z_]], not[subclass[x_, cart[y_, y_]]]] := True
```

Theorem.

```
In[83]:= Map[implies[member[y, z], not[#]] &,
           SubstTest[and, implies[p1, p5], implies[and[p3, p5], p2], implies[p1, p4],
            implies[and[p1, p2, p3, p4], p6], not[implies[and[p1, p3], p6]],
            {p1 → member[y, BIJ], p2 → member[x, V], p3 → subclass[x, cart[domain[y], domain[y]]],
             p4 → member[composite[y, x, inverse[y]], V], p5 → member[domain[y], V],
             p6 → member[pair[x, composite[y, x, inverse[y]]], SIMILAR]}]] // Reverse

Out[83]= or[member[pair[x, composite[y, x, inverse[y]]], SIMILAR],
          not[FUNCTION[y]], not[FUNCTION[inverse[y]]], not[member[y, z]],
          not[subclass[x, cart[domain[y], domain[y]]]]] == True

In[84]:= or[member[pair[x_, composite[y_, x_, inverse[y_]]], SIMILAR],
          not[FUNCTION[inverse[y_]]], not[FUNCTION[y_]], not[member[y_, z_]],
          not[subclass[x_, cart[domain[y_], domain[y_]]]]] := True
```

partial orders

In this section it is shown that any relation similar to a partial order is a partial order, and any partial order is similar to a restriction of the subset relation.

Lemma. An inclusion in one direction.

```
In[85]:= SubstTest[implies, subclass[u, v],
           subclass[image[u, w], image[v, w]], {u -> SIMILAR, v -> composite[IMG,
             id[cart[image[CROSS, id[BIJ]], V]], inverse[SECOND]], w → PO}] // Reverse

Out[85]= subclass[image[SIMILAR, PO], PO] == True

In[86]:= % /. Equal→ SetDelayed
```

Theorem. Any relation similar to a partial order is a partial order.

```
In[87]:= SubstTest[and, subclass[u, v], subclass[v, u], {u -> image[SIMILAR, PO], v → PO}]

Out[87]= equal[PO, image[SIMILAR, PO]] == True

In[88]:= image[SIMILAR, PO] := PO
```

The result derived in the preceding section is applied to the case of a canonical bijection.

Lemma.

```
In[89]:= SubstTest[or, member[pair[u, composite[v, u, inverse[v]]], SIMILAR],
           not[FUNCTION[v]], not[FUNCTION[inverse[v]]], not[member[v, V]],
           not[subclass[u, cart[domain[v], domain[v]]]],
           {u → po[setpart[x]], v → APPLY[VS, inverse[po[setpart[x]]]]}] // Reverse

Out[89]= member[pair[po[setpart[x]],
            composite[id[image[VERTSECT[inverse[po[setpart[x]]]], fix[po[setpart[x]]]]], S, id[
              image[VERTSECT[inverse[po[setpart[x]]]], fix[po[setpart[x]]]]]], SIMILAR] == True

In[90]:= (% /. x → x_) /. Equal→ SetDelayed
```

The class of all restrictions of **x** to cartesian squares is **image[IMAGE[id[x]], image[CART,Id]]**.

Lemma.

```
In[91]:= Map[implies[member[x, z], #] &,
           SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
            {t → IMAGE[id[y]], u → set[cart[x, x]], v → image[CART, Id]}]] // Reverse

Out[91]= or[member[composite[id[x], y, id[x]], image[IMAGE[id[y]], image[CART, Id]]],
           not[member[x, z]]] == True

In[92]:= or[member[composite[id[x_], y_, id[x_]], image[IMAGE[id[y_]], image[CART, Id]]],
           not[member[x_, z_]]] := True
```

Lemma. A special case of interest.

```
In[93]:= SubstTest[implies, member[t, V],
           member[composite[id[t], y, id[t]], image[IMAGE[id[y]], image[CART, Id]]], {y → S,
            t -> image[VERTSECT[inverse[po[setpart[x]]]], fix[po[setpart[x]]]]}] // Reverse

Out[93]= member[composite[id[image[VERTSECT[inverse[po[setpart[x]]]], fix[po[setpart[x]]]]],
           S, id[image[VERTSECT[inverse[po[setpart[x]]]], fix[po[setpart[x]]]]]],
          image[IMAGE[id[S]], image[CART, Id]]] == True

In[94]:= (% /. x → x_) /. Equal→ SetDelayed
```

Lemma. Any small partial order is similar to restriction of **S** to a cartesian square.

```
In[95]:= SubstTest[implies, and[member[pair[u, v], composite[Id, w]], member[v, t]],
           member[u, image[inverse[w], t]],
           {t → image[IMAGE[id[S]], image[CART, Id]], u → po[setpart[x]],
            v -> composite[id[image[VERTSECT[inverse[po[setpart[x]]]], fix[po[setpart[x]]]]],
              S, id[image[VERTSECT[inverse[po[setpart[x]]]], fix[po[setpart[x]]]]], w →
             SIMILAR}] // Reverse

Out[95]= member[po[setpart[x]], image[SIMILAR, image[IMAGE[id[S]], image[CART, Id]]]] == True

In[96]:= (% /. x → x_) /. Equal→ SetDelayed
```

Lemma. An inclusion obtained by eliminating the variable **x** in the preceding lemma.

```
In[97]:= Map[or[subclass[PO, image[SIMILAR, image[IMAGE[id[S]], image[CART, Id]]]], empty[#]] &,
          SubstTest[reify, x, dif[set[po[setpart[x]]], t],
           t -> image[SIMILAR, image[IMAGE[id[S]], image[CART, Id]]]]]

Out[97]= subclass[PO, image[SIMILAR, image[IMAGE[id[S]], image[CART, Id]]]] == True

In[98]:= % /. Equal→ SetDelayed
```

Lemma. The opposite inclusion.

```
In[99]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
           {t → SIMILAR, u -> image[IMAGE[id[S]], image[CART, Id]], v → PO}] // Reverse

Out[99]= subclass[image[SIMILAR, image[IMAGE[id[S]], image[CART, Id]]], PO] == True

In[100]:=
        % /. Equal→ SetDelayed
```

Theorem. Every partial order is similar to a restriction of the subset relation **S**.

```
In[101]:=
        SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> image[SIMILAR, image[IMAGE[id[S]], image[CART, Id]]], v -> PO}]

Out[101]=
        equal[PO, image[SIMILAR, image[IMAGE[id[S]], image[CART, Id]]]] == True

In[102]:=
        image[SIMILAR, image[IMAGE[id[S]], image[CART, Id]]] := PO
```