intersections and unions of reflexive relations

Johan G. F. Belinfante
2003 September 22

```
In[1]:= << goedel52.s95; << tools.m

        :Package Title: goedel52.s95      2003 September 18 at 7:45 a.m.

        It is now:  2003 Sep 22 at 8:58

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 August 9

        weightlimit = 40
```

summary

In this notebook it is shown that a relation is reflexive if and only if it is the union of its reflexive subsets. A number of other facts about reflexive relations are derived which are needed for the proof, plus some applications. One of these byproducts is a formula for **CORE[RFX]**. Although the details are somewhat technical, the underlying idea is extremely simple. One can think of a relation as a directed graph, with each edge of the graph representing one of the ordered pairs that belong to the relation. A relation is reflexive if there is a self−loop at each vertex. To obtain the largest reflexive relation contained in a given relation one simply needs to remove all edges that start or end with a vertex that fails to have a self−loop. Although it is not addressed in this notebook, there should be a similar result for **HULL[RFX]**. To obtain the smallest reflexive relation that contains a given relation, one needs only to add a self−loop at any vertex that lacks one.

preliminaries

The predicate **REFLEXIVE** is defined in the **GOEDEL** program using a single literal:

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

Out[2]= subclass[x, cart[fix[x], fix[x]]]
```

In this section a rewrite rule is derived to enable the **GOEDEL** program to recognize an equivalent formulation with three literals which is needed in the sequel. The basic idea for combining three properties into a single condition is this:

```
In[3]:= Map[or[#, not[subclass[domain[x], fix[x]]], not[subclass[range[x], fix[x]]]] &,
          SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
            {u -> x, v -> cart[domain[x], range[x]], w -> cart[fix[x], fix[x]]}]]

Out[3]= or[not[subclass[x, cart[domain[x], range[x]]]], not[subclass[domain[x], fix[x]]],
          not[subclass[range[x], fix[x]]], subclass[x, cart[fix[x], fix[x]]]] == True

In[4]:= (% /. x -> x_) /. Equal -> SetDelayed
```

The preceding statement can be cleaned up a bit:

```
In[5]:= Map[not, SubstTest[and, implies[p1, p4],
          implies[and[p2, p3, p4], p5], not[implies[and[p1, p2, p3], p5]],
            {p1 -> subclass[x, cart[V, V]], p2 -> subclass[domain[x], fix[x]],
           p3 -> subclass[range[x], fix[x]],
                p4 -> subclass[x, cart[domain[x], range[x]]], p5 -> REFLEXIVE[x]}]]

Out[5]= or[not[subclass[x, cart[V, V]]], not[subclass[domain[x], fix[x]]],
          not[subclass[range[x], fix[x]]], subclass[x, cart[fix[x], fix[x]]]] == True

In[6]:= (% /. x -> x_) /. Equal -> SetDelayed
```

The converse also holds:

```
In[7]:= implies[REFLEXIVE[x], and[subclass[domain[x], fix[x]],
           subclass[range[x], fix[x]], subclass[x, cart[V, V]]]] // NotNotTest

Out[7]= or[and[subclass[x, cart[V, V]], subclass[domain[x], fix[x]],
           subclass[range[x], fix[x]]], not[subclass[x, cart[fix[x], fix[x]]]]] == True

In[8]:= (% /. {x -> x_}) /. Equal -> SetDelayed
```

The above results can be summarized as a logical equivalence:

```
In[9]:= equiv[REFLEXIVE[x],
          and[subclass[domain[x], fix[x]], subclass[range[x], fix[x]], subclass[x, cart[V, V]]]]

Out[9]= True
```

This equivalence is made into a new rewrite rule:

```
In[10]:= and[subclass[x_, cart[V, V]], subclass[domain[x_], fix[x_]],
           subclass[range[x_], fix[x_]]] := subclass[x, cart[fix[x], fix[x]]]
```

equality substitution rule for REFLEXIVE

One of the ingredients in the derivation of the main result is an equality substitution rule for the predicate **REFLEXIVE**. First a lemma:

```
In[11]:= SubstTest[implies, and[subclass[x, u], equal[u, v]], subclass[x, v],
             {u -> cart[y, y], v -> cart[z, z]}]

Out[11]= or[not[equal[y, z]], not[subclass[x, cart[y, y]]], subclass[x, cart[z, z]]] == True

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

The desired substitution rule is derived as follows:

```
In[13]:= Map[not, SubstTest[and, implies[p1, p3],
           implies[and[p2, p3], p4], implies[and[p1, p4], p5],
                 not[implies[and[p1, p2], p5]],
             {p1 -> equal[x, y], p2 -> REFLEXIVE[x], p3 -> equal[fix[x], fix[y]],
                 p4 -> subclass[x, cart[fix[y], fix[y]]], p5 -> REFLEXIVE[y]}]]

Out[13]= or[not[equal[x, y]], not[subclass[x, cart[fix[x], fix[x]]]],
           subclass[y, cart[fix[y], fix[y]]]] == True

In[14]:= or[not[equal[x_, y_]], not[subclass[x_, cart[fix[x_], fix[x_]]]],
           subclass[y_, cart[fix[y_], fix[y_]]]] := True
```

Restatement:

```
In[15]:= implies[and[equal[x, y], REFLEXIVE[x]], REFLEXIVE[y]]

Out[15]= True
```

binary unions and intersections of reflexive relations

A double negation suffices for the case of binary unions, but this does not suffice for binary intersections

```
In[16]:= implies[and[REFLEXIVE[x], REFLEXIVE[y]], REFLEXIVE[union[x, y]]] // not // not

Out[16]= True
```

The following new rewrite rule is added:

```
In[17]:= SubstTest[subclass, u, intersection[s, t], {s -> cart[v, x], t -> cart[w, y]}]

Out[17]= subclass[u, cart[intersection[v, w], intersection[x, y]]] ==
          and[subclass[u, cart[v, x]], subclass[u, cart[w, y]]]

In[18]:= subclass[u_, cart[intersection[v_, w_], intersection[x_, y_]]] :=
          and[subclass[u, cart[v, x]], subclass[u, cart[w, y]]]
```

This works fine:

```
In[19]:= implies[and[REFLEXIVE[x], REFLEXIVE[y]], REFLEXIVE[intersection[x, y]]]

Out[19]= True
```

arbitrary unions of reflexive relations

In this section it is shown that the union (sum class) of any collection of reflexive relations is a reflexive relation. For the case that the collections are sets, there already is an existing rule:

```
In[20]:= Uclosure[RFX]

Out[20]= RFX
```

The result obtained below generalizes this result by removing the restriction to sets. The main idea of the derivation presented here is to obtain the desired result by specializing a general theorem about invariant subsets. The class **RFX** of all (small) reflexive relations can be written as the class of invariant subsets for a suitable relation. There is more than one way to do this; here is one way:

```
In[21]:= invar[union[composite[DUP, union[FIRST, SECOND]], cart[complement[cart[V, V]], V]]]

Out[21]= RFX
```

This observation can be used to deduce many results about **RFX** as corollaries of general theorems about **invar[x]**. For example, one can deduce a variable−free rule about binary unions

```
In[22]:= SubstTest[image, CUP, cart[invar[x], invar[x]],
          x -> union[composite[DUP, union[FIRST, SECOND]], cart[complement[cart[V, V]], V]]]

Out[22]= image[CUP, cart[RFX, RFX]] == RFX

In[23]:= image[CUP, cart[RFX, RFX]] := RFX
```

For the case of arbitrary unions, one needs to consider invariant subclasses as well as invariant subsets. There is no **x** whose invariant subclasses are all the reflexive relations. The reason is that for any **x**, the universal class **V** is an invariant subclass, but **V** fails to be a reflexive relation. For the particular relation **x** used above, the universal class is the only exception; all other invariant subclasses are reflexive relations. To obtain a clean result for arbitrary unions, one needs two lemmas. The first lemma uses double negation to combine two rewrite rules into one; this is needed to rule out the possibility that **U[x] = V**.

```
In[24]:= implies[subclass[x, invar[y]],
           and[invariant[y, U[x]], subclass[U[x], U[invar[y]]]]] // NotNotTest

Out[24]= or[and[subclass[image[y, U[x]], U[x]], subclass[U[x], U[invar[y]]]],
           not[subclass[x, invar[y]]]] == True

In[25]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

The second lemma gets rid of a redundant literal that would otherwise remain in the conclusion:

```
In[26]:= equiv[and[subclass[x, cart[y, z]], subclass[x, cart[V, V]]], subclass[x, cart[y, z]]]

Out[26]= True

In[27]:= and[subclass[x_, cart[V, V]], subclass[x_, cart[y_, z_]]] := subclass[x, cart[y, z]]
```

The main result of this section now emerges:

```
In[28]:= SubstTest[implies, subclass[x, invar[y]],
          and[invariant[y, U[x]], subclass[U[x], U[invar[y]]]],
            y -> union[composite[DUP, union[FIRST, SECOND]], cart[complement[cart[V, V]], V]]]

Out[28]= or[not[subclass[x, RFX]], subclass[U[x], cart[fix[U[x]], fix[U[x]]]]] == True

In[29]:= (% /. x -> x_) /. Equal -> SetDelayed
```

Restatement:

```
In[30]:= implies[subclass[x, RFX], REFLEXIVE[U[x]]]

Out[30]= True
```

arbitrary intersections

In this section a rule is derived for the intersection of an arbitrary, but not empty, class of reflexive relations. The case of an empty collection needs to be excluded because **A[0]=V** is not a subclass of **cart[V,V]**.

```
In[31]:= Map[or[equal[0, x], #] &, SubstTest[implies, subclass[x, invar[y]], invariant[y, A[x]],
           y -> union[composite[DUP, union[FIRST, SECOND]], cart[complement[cart[V, V]], V]]]]

Out[31]= or[equal[0, x], not[subclass[x, RFX]],
           subclass[A[x], cart[fix[A[x]], fix[A[x]]]]] == True
```

```
In[32]:= (% /. x -> x_) /. Equal -> SetDelayed
```

Restatement:

```
In[33]:= implies[and[not[equal[0, x]], subclass[x, RFX]], REFLEXIVE[A[x]]]

Out[33]= True
```

equality substitution for core

In this section another equality substitution rule is derived, but it is not used further in this notebook. Instead, a monotonicity argument will be used. The idea here is to drive the equality rule from the monotonicity rule. A temporary abbreviation is introduced:

```
In[34]:= core[x_, y_] := U[intersection[x, P[y]]]
```

The equality in the hypothesis can be handled with an **AssertTest**.

```
In[35]:= implies[equal[x, y], subclass[core[z, x], core[z, y]]] // AssertTest

Out[35]= or[not[equal[x, y]],
           subclass[U[intersection[z, P[x]]], U[intersection[z, P[y]]]]] == True
```

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

The equality in the conclusion follows by combining an inclusion and its reverse. In this case the reverse inclusion is essentially the same as the inclusion itself due to symmetry of the equality hypothesis.

```
In[37]:= SubstTest[and, implies[p, subclass[u, v]], implies[p, subclass[v, u]],
             {p -> equal[y, z], u -> core[x, y], v -> core[x, z]}] // Reverse

Out[37]= or[equal[U[intersection[x, P[y]]], U[intersection[x, P[z]]]], not[equal[y, z]]] == True
```

```
In[38]:= or[equal[U[intersection[x_, P[y_]]], U[intersection[x_, P[z_]]]],
           not[equal[y_, z_]]] := True
```

el

In this section an arbitrary reflexive relation is written as the union of its reflexive subsets. For this purpose it is useful to introduce the smallest reflexive relations, which are three−point sets shaped like the letter **L**.

```
In[39]:= member[union[cart[singleton[x], singleton[y]], id[pairset[x, y]]], RFX]

Out[39]= True
```

A temporary name for these sets is introduced:

```
In[40]:= el[x_, y_] := union[cart[singleton[x], singleton[y]], id[pairset[x, y]]]
```

```
In[41]:= SubstTest[implies, and[member[w, x], subclass[x, y]], member[w, y],
           {w -> pair[u, v], y -> cart[fix[x], fix[x]]}]

Out[41]= or[and[member[u, fix[x]], member[v, fix[x]]],
           not[member[pair[u, v], x]], not[subclass[x, cart[fix[x], fix[x]]]]] == True

In[42]:= (% /. {u -> u_, v -> v_, x -> x_}) /. Equal -> SetDelayed

In[43]:= Map[implies[and[REFLEXIVE[x], member[pair[u, v], x]], #] &,
          subclass[el[u, v], x] // AssertTest]

Out[43]= or[not[member[pair[u, v], x]],
           not[subclass[x, cart[fix[x], fix[x]]]], subclass[pairset[u, v], fix[x]]] == True

In[44]:= (% /. {u -> u_, v -> v_, x -> x_}) /. Equal -> SetDelayed

In[45]:= 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[RFX, P[x]]}]]

Out[45]= or[member[pair[u, v], U[intersection[RFX, P[x]]]], not[member[u, V]], not[member[v, V]],
           not[member[pair[u, v], x]], not[subclass[pairset[u, v], fix[x]]]] == True

In[46]:= (% /. {u -> u_, v -> v_, x -> x_}) /. Equal -> SetDelayed

In[47]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[and[p1, p2], p4],
           implies[p4, p5], implies[p4, p6], implies[and[p2, p3, p5, p6], p7],
                 not[implies[and[p1, p2], p7]],
             {p1 -> REFLEXIVE[x], p2 -> member[pair[u, v], x], p3 ->
             subclass[pairset[u, v], fix[x]], p4 -> and[member[u, fix[x]], member[v, fix[x]]],
                 p5 -> member[u, V], p6 -> member[v, V],
              p7 -> member[pair[u, v], U[intersection[RFX, P[x]]]]}]]

Out[47]= or[member[pair[u, v], U[intersection[RFX, P[x]]]],
           not[member[pair[u, v], x]], not[subclass[x, cart[fix[x], fix[x]]]]] == True
```

The set variables **u** and **v** can be eliminated:

```
In[48]:= Map[equal[0, composite[Id, complement[class[pair[u, v], #]]]] &, %] // MapNotNot

Out[48]= or[not[subclass[x, cart[fix[x], fix[x]]]],
           subclass[composite[Id, x], U[intersection[RFX, P[x]]]]] == True

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

A cleaner version of this result can be obtained:

```
In[50]:= SubstTest[implies, and[equal[x, y], subclass[y, z]], subclass[x, z],
           {y -> composite[Id, x], z -> U[intersection[RFX, P[x]]]}]

Out[50]= or[not[subclass[x, cart[V, V]]],
           not[subclass[composite[Id, x], U[intersection[RFX, P[x]]]]],
           subclass[x, U[intersection[RFX, P[x]]]]] == True

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

```
In[52]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p1, p3], implies[and[p2, p3], p4], not[implies[p1, p4]],
         {p1 -> REFLEXIVE[x], p2 -> subclass[x, cart[V, V]],
            p3 -> subclass[composite[Id, x], U[intersection[RFX, P[x]]]],
                 p4 -> subclass[x, U[intersection[RFX, P[x]]]]}]]

Out[52]= or[not[subclass[x, cart[fix[x], fix[x]]]],
           subclass[x, U[intersection[RFX, P[x]]]]] == True

In[53]:= (% /. x -> x_) /. Equal -> SetDelayed
```

Next, this inclusion can be strengthened to an equation:

```
In[54]:= SubstTest[and, implies[p, subclass[x, y]], subclass[y, x],
             {p -> REFLEXIVE[x], y -> U[intersection[RFX, P[x]]]}] // Reverse

Out[54]= or[equal[x, U[intersection[RFX, P[x]]]], not[subclass[x, cart[fix[x], fix[x]]]]] == True
```

This says that any reflexive relation is the union of its reflexive subsets.

```
In[55]:= or[equal[x_, U[intersection[RFX, P[x_]]]],
           not[subclass[x_, cart[fix[x_], fix[x_]]]]] := True
```

The converse is also true.

```
In[56]:= SubstTest[implies, subclass[y, RFX], REFLEXIVE[U[y]], y -> intersection[RFX, P[x]]]

Out[56]= subclass[U[intersection[RFX, P[x]]],
           cart[fix[U[intersection[RFX, P[x]]]], fix[U[intersection[RFX, P[x]]]]]] == True

In[57]:= (% /. x -> x_) /. Equal -> SetDelayed
```

The equality substitution rule for **REFLEXIVE** is needed at this point.

```
In[58]:= SubstTest[implies, and[equal[x, y], REFLEXIVE[y]], REFLEXIVE[x], y -> core[RFX, x]]

Out[58]= or[not[equal[x, U[intersection[RFX, P[x]]]]], subclass[x, cart[fix[x], fix[x]]]] == True

In[59]:= (% /. x -> x_) /. Equal -> SetDelayed
```

The two implications yield a logical equivalence:

```
In[60]:= equiv[equal[x, U[intersection[RFX, P[x]]]], subclass[x, cart[fix[x], fix[x]]]]

Out[60]= True
```

This yields a temporary rewrite rule:

```
In[61]:= equal[x_, U[intersection[RFX, P[x_]]]] := subclass[x, cart[fix[x], fix[x]]]
```

Restatement:

```
In[62]:= equal[x, core[RFX, x]] == REFLEXIVE[x]

Out[62]= True
```

a formula for core[RFX, x]

The goal will be to show that the reflexive core is identical to the following expression for which a temporary name is introduced:

```
In[63]:= RFXpart[x_] := intersection[x, cart[fix[x], fix[x]]]
```

The **GOEDEL** program now recognizes this to be true because **RFXpart[x]** is a reflexive relation.

```
In[64]:= equal[core[RFX, RFXpart[x]], RFXpart[x]]

Out[64]= True
```

There is as yet no rule that rewrites the left side of the equation to the right side. This can be seen by replacing **equal** with **Equal**.

```
In[65]:= Equal[core[RFX, RFXpart[x]], RFXpart[x]]

Out[65]= U[intersection[RFX, P[composite[id[fix[x]], x, id[fix[x]]]]]] ==
          composite[id[fix[x]], x, id[fix[x]]]
```

Since these expressions are in fact equal, it is justified to add this rewrite rule:

```
In[66]:= U[intersection[RFX, P[composite[id[fix[x_]], x_, id[fix[x_]]]]]] :=
          composite[id[fix[x]], x, id[fix[x]]]
```

Lemma.

```
In[67]:= equal[x, RFXpart[x]] // AssertTest

Out[67]= equal[x, composite[id[fix[x]], x, id[fix[x]]]] == subclass[x, cart[fix[x], fix[x]]]

In[68]:= equal[x_, composite[id[fix[x_]], x_, id[fix[x_]]]] := subclass[x, cart[fix[x], fix[x]]]
```

A similar result holds in the opposite direction:

```
In[69]:= equal[RFXpart[core[RFX, x]], core[RFX, x]]

Out[69]= True

In[70]:= Equal[RFXpart[core[RFX, x]], core[RFX, x]]

Out[70]= composite[id[fix[U[intersection[RFX, P[x]]]]], U[intersection[RFX, P[x]]],
           id[fix[U[intersection[RFX, P[x]]]]]] == U[intersection[RFX, P[x]]]

In[71]:= composite[id[fix[U[intersection[RFX, P[x_]]]]], U[intersection[RFX, P[x_]]],
           id[fix[U[intersection[RFX, P[x_]]]]]] := U[intersection[RFX, P[x]]]
```

applying monotonicity

A monotonicity argument is used to complete the proof that the **core[RFX,x]** and **RFXpart[x]** are identical. In one direction, one has:

```
In[72]:= SubstTest[implies, subclass[y, x], subclass[core[RFX, y], core[RFX, x]], y → RFXpart[x]]

Out[72]= subclass[composite[id[fix[x]], x, id[fix[x]]], U[intersection[RFX, P[x]]]] == True

In[73]:= (% /. x -> x_) /. Equal -> SetDelayed
```

The other direction requires extra work:

```
In[74]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
          {u → RFXpart[y], v -> cart[fix[y], fix[y]], w → cart[fix[x], fix[x]]}]

Out[74]= or[and[subclass[intersection[fix[y], image[y, fix[y]]], fix[x]],
             subclass[intersection[fix[y], image[inverse[y], fix[y]]], fix[x]]],
           not[subclass[fix[y], fix[x]]]] == True

In[75]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed

In[76]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
            not[implies[p1, p3]], {p1 → subclass[y, x], p2 → subclass[fix[y], fix[x]],
             p3 -> and[subclass[intersection[fix[y], image[y, fix[y]]], fix[x]],
               subclass[intersection[fix[y], image[inverse[y], fix[y]]], fix[x]]]}]]

Out[76]= or[and[subclass[intersection[fix[y], image[y, fix[y]]], fix[x]],
             subclass[intersection[fix[y], image[inverse[y], fix[y]]], fix[x]]],
           not[subclass[y, x]]] == True

In[77]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed

In[78]:= implies[subclass[y, x], subclass[RFXpart[y], RFXpart[x]]] // NotNotTest

Out[78]= or[and[subclass[composite[id[fix[y]], y, id[fix[y]]], x],
             subclass[intersection[fix[y], image[y, fix[y]]], fix[x]],
             subclass[intersection[fix[y], image[inverse[y], fix[y]]], fix[x]]],
           not[subclass[y, x]]] == True

In[79]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed

In[80]:= SubstTest[subclass, RFXpart[y], RFXpart[x], y → core[RFX, x]] // Reverse

Out[80]= and[subclass[intersection[fix[U[intersection[RFX, P[x]]]],
             image[inverse[U[intersection[RFX, P[x]]]], fix[U[intersection[RFX, P[x]]]]]],
           fix[x]], subclass[intersection[fix[U[intersection[RFX, P[x]]]],
             image[U[intersection[RFX, P[x]]], fix[U[intersection[RFX, P[x]]]]], fix[x]]] ==
         subclass[U[intersection[RFX, P[x]]], cart[fix[x], fix[x]]]

In[81]:= and[subclass[intersection[fix[U[intersection[RFX, P[x_]]]],
             image[inverse[U[intersection[RFX, P[x_]]]], fix[U[intersection[RFX, P[x_]]]]],
           fix[x_]], subclass[intersection[fix[U[intersection[RFX, P[x_]]]],
             image[U[intersection[RFX, P[x_]]], fix[U[intersection[RFX, P[x_]]]]], fix[x_]]] :=
         subclass[U[intersection[RFX, P[x]]], cart[fix[x], fix[x]]]

In[82]:= SubstTest[implies, subclass[y, x], subclass[RFXpart[y], RFXpart[x]], y → core[RFX, x]]

Out[82]= subclass[U[intersection[RFX, P[x]]], cart[fix[x], fix[x]]] == True

In[83]:= (% /. x → x_) /. Equal → SetDelayed
```

It remains to put the two pieces together:

```
In[84]:= SubstTest[and, subclass[u, v],
           subclass[v, u], {u → RFXpart[x], v → core[RFX, x]}] // Reverse

Out[84]= equal[composite[id[fix[x]], x, id[fix[x]]], U[intersection[RFX, P[x]]]] == True

In[85]:= U[intersection[RFX, P[x_]]] := composite[id[fix[x]], x, id[fix[x]]]
```

functional formulation

The formula derived in the preceding section has a functional counterpart. To derive it, a lemma is needed:

```
In[86]:= composite[CAP, cross[Id, composite[CART, DUP, FIX]], DUP] // VSNormality // Reverse

Out[86]= intersection[
           composite[S, COMPOSE, intersection[composite[inverse[SECOND], IMAGE[id[Id]]],
             composite[inverse[FIRST], COMPOSE, intersection[composite[inverse[FIRST],
                IMAGE[id[Id]]], composite[inverse[SECOND], IMAGE[id[cart[V, V]]]]]]]],
           composite[inverse[S], CART, DUP, IMAGE[inverse[DUP]]], inverse[S]] ==
         composite[CAP, id[composite[CART, DUP, IMAGE[inverse[DUP]]]], inverse[FIRST]]

In[87]:= % /. Equal -> SetDelayed
```

The result now follows:

```
In[88]:= CORE[RFX] // VSNormality // Reverse

Out[88]= composite[CAP, id[composite[CART, DUP, IMAGE[inverse[DUP]]]], inverse[FIRST]] ==
          CORE[RFX]

In[89]:= composite[CAP, id[composite[CART, DUP, IMAGE[inverse[DUP]]]], inverse[FIRST]] :=
          CORE[RFX]
```

A corollary:

```
In[90]:= Assoc[FIX, CAP,
          composite[id[composite[CART, DUP, IMAGE[inverse[DUP]]]], inverse[FIRST]]]

Out[90]= composite[IMAGE[inverse[DUP]], CORE[RFX]] == IMAGE[inverse[DUP]]

In[91]:= composite[IMAGE[inverse[DUP]], CORE[RFX]] := IMAGE[inverse[DUP]]
```