Theorem FS−CUP

Johan G. F. Belinfante
2003 October 8

```
In[1]:= << goedel52.t00; << tools.m

        :Package Title: goedel52.t00      2003 October 7 at 4:05 p.m.

        It is now:  2003 Oct 8 at 9:4

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 September 29

        weightlimit = 40
```

summary

The current rewrite rule for unions of functions should be replaced for two reasons. In the first place, the current rule rewrites a simple fact to a complicated result. In the second place, this complicated result contains redundant information. Replacement rules are derived in this notebook.

Theorem **FS−CUP** proved 2000 March 9 using **Otter** asserts that the sum class of a collection of pairwise compatible functions is a function, with **image[inverse[CUP], FUNS]** as the compatibility relation. This result is rederived in this notebook using standard techniques available in the **GOEDEL** program similar to those used in the **Otter** proof, but of course, the **Otter** proof was automatic, whereas the present derivation is hand−guided. The converse also holds.

removing a rewrite rule for FUNCTION[union[x,y]]

Thr **GOEDEL** program currently contains this symmetric rewrite rule:

```
In[2]:= FUNCTION[union[x, y]]

Out[2]= and[equal[0, fix[composite[x, inverse[y], Di]]],
          equal[0, fix[composite[y, inverse[x], Di]]], FUNCTION[x], FUNCTION[y]]
```

The rule is correct, but produces redundant information. The two **fix** assertions are logically equivalent. This is most easily seen as follows:

```
In[3]:= subclass[composite[x, inverse[y]], Id] // AssertTest // Reverse

Out[3]= equal[0, fix[composite[inverse[y], Di, x]]] == subclass[composite[x, inverse[y]], Id]

In[4]:= equal[0, fix[composite[inverse[y_], Di, x_]]] := subclass[composite[x, inverse[y]], Id]

In[5]:= SubstTest[implies, equal[0, fix[composite[u, v]]],
          equal[0, fix[composite[v, u]]], {u -> x, v -> composite[inverse[y], Di]}]

Out[5]= or[not[equal[0, fix[composite[x, inverse[y], Di]]]],
           subclass[composite[x, inverse[y]], Id]] == True
```

```
In[6]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed

In[7]:= SubstTest[implies, equal[0, fix[composite[u, v]]],
         equal[0, fix[composite[v, u]]], {v -> x, u -> composite[inverse[y], Di]}]

Out[7]= or[equal[0, fix[composite[x, inverse[y], Di]]],
          not[subclass[composite[x, inverse[y]], Id]]] == True

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

Note that:

```
In[9]:= equiv[equal[0, fix[composite[x, inverse[y], Di]]],
         subclass[composite[x, inverse[y]], Id]]

Out[9]= True
```

This justifies adding this (temporary) rewrite rule:

```
In[10]:= equal[0, fix[composite[x_, inverse[y_], Di]]] := subclass[composite[x, inverse[y]], Id]
```

With these new rules, one finds:

```
In[11]:= FUNCTION[union[x, y]]

Out[11]= and[FUNCTION[x], FUNCTION[y], subclass[composite[x, inverse[y]], Id],
          subclass[composite[y, inverse[x]], Id]]
```

The two **subclass** conditions are equivalent:

```
In[12]:= Map[implies[subclass[composite[x, inverse[y]], Id], #] &,
          SubstTest[subclass, inverse[u], inverse[v], {u -> composite[x, inverse[y]], v -> Id}]]

Out[12]= or[not[subclass[composite[x, inverse[y]], Id]],
           subclass[composite[y, inverse[x]], Id]] == True

In[13]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

Thus:

```
In[14]:= equiv[subclass[composite[x, inverse[y]], Id], subclass[composite[y, inverse[x]], Id]]

Out[14]= True
```

Accordingly, one could simplify the rule for **FUNCTION[x,y]** as follows:

```
In[15]:= equiv[FUNCTION[union[x, y]],
          and[FUNCTION[x], FUNCTION[y], subclass[composite[x, inverse[y]], Id]]]

Out[15]= True
```

Rather than do this, which would produce unsymmetric formulas, it seems more appropriate to remove the rule altogether and replace it with a set of implications.

```
In[16]:= FUNCTION[union[x_, y_]] =.
```

To facilitate the derivation of replacement rules, it is convenient to turn off some flags:

```
In[17]:= simplify = False; cond = False;
```

Application of **AssertTest** yields some replacement rules:

```
In[18]:= implies[FUNCTION[union[x, y]], FUNCTION[x]] // AssertTest

Out[18]= or[FUNCTION[x], not[FUNCTION[union[x, y]]]] == True

In[19]:= or[FUNCTION[x_], not[FUNCTION[union[x_, y_]]]] := True

In[20]:= implies[FUNCTION[union[x, y]], subclass[composite[x, inverse[y]], Id]] // AssertTest

Out[20]= or[not[FUNCTION[union[x, y]]], subclass[composite[x, inverse[y]], Id]] == True

In[21]:= or[not[FUNCTION[union[x_, y_]]], subclass[composite[x_, inverse[y_]], Id]] := True
```

This yields half of the rule removed:

```
In[22]:= implies[FUNCTION[union[x, y]],
           and[FUNCTION[x], FUNCTION[y], subclass[composite[x, inverse[y]], Id],
            subclass[composite[y, inverse[x]], Id]]] // NotNotTest

Out[22]= or[and[FUNCTION[x], FUNCTION[y], subclass[composite[x, inverse[y]], Id],
            subclass[composite[y, inverse[x]], Id]], not[FUNCTION[union[x, y]]]] == True
```

In the opposite direction, one has this stronger result:

```
In[23]:= implies[and[FUNCTION[x], FUNCTION[y], subclass[composite[x, inverse[y]], Id]],
           FUNCTION[union[x, y]]] // AssertTest

Out[23]= or[FUNCTION[union[x, y]], not[FUNCTION[x]],
           not[FUNCTION[y]], not[subclass[composite[x, inverse[y]], Id]]] == True

In[24]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

an equivalent formulation

In this section an alternative criterion is derived for a union of two functions to be a function which is often more convenient for applications. First a rather technical lemma:

```
In[25]:= equal[x, union[composite[Id, x], composite[y, id[domain[x]]]]] // AssertTest

Out[25]= equal[x, union[composite[Id, x], composite[y, id[domain[x]]]]] ==
          and[subclass[x, cart[V, V]], subclass[composite[y, id[domain[x]]], x]]

In[26]:= equal[x_, union[composite[Id, x_], composite[y_, id[domain[x_]]]]] :=
          and[subclass[x, cart[V, V]], subclass[composite[y, id[domain[x]]], x]]
```

The necessity of the new condition is easily established:

```
In[27]:= Map[or[subclass[composite[x, id[domain[y]]], y], #] &,
          SubstTest[implies, and[subclass[y, z], FUNCTION[z]],
           equal[y, composite[z, id[domain[y]]]], z -> union[x, y]]]

Out[27]= or[not[FUNCTION[union[x, y]]], subclass[composite[x, id[domain[y]]], y]] == True

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

Lemma.

```
In[29]:= SubstTest[implies, subclass[u, v], subclass[composite[x, u], composite[x, v]],
           {u -> id[domain[y]], v -> composite[inverse[y], y]}]

Out[29]= subclass[composite[x, id[domain[y]]], composite[x, inverse[y], y]] == True

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

The old condition implies the new one:

```
In[31]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 -> subclass[composite[x, inverse[y]], Id],
            p2 -> subclass[composite[x, inverse[y], y], y],
                 p3 -> subclass[composite[x, id[domain[y]]], y]}]]

Out[31]= or[not[subclass[composite[x, inverse[y]], Id]],
           subclass[composite[x, id[domain[y]]], y]] == True

In[32]:= or[not[subclass[composite[x_, inverse[y_]], Id]],
           subclass[composite[x_, id[domain[y_]]], y_]] := True
```

In the reverse direction, one needs to know that **y** is a function.

```
In[33]:= SubstTest[implies, subclass[u, v], subclass[composite[u, w], composite[v, w]],
           {u -> composite[x, id[domain[y]]], v -> y, w -> inverse[y]}]

Out[33]= or[not[subclass[composite[x, id[domain[y]]], y]],
           subclass[composite[x, inverse[y]], composite[y, inverse[y]]]] == True

In[34]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed

In[35]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u -> composite[x, inverse[y]], v -> composite[y, inverse[y]], w -> Id}]

Out[35]= or[not[FUNCTION[composite[Id, y]]],
           not[subclass[composite[x, inverse[y]], composite[y, inverse[y]]]],
           subclass[composite[x, inverse[y]], Id]] == True

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

In[37]:= Map[not, SubstTest[and, implies[p1, p3], implies[p2, p4],
           implies[and[p3, p4], p5], not[implies[and[p1, p2], p5]],
             {p1 -> subclass[composite[x, id[domain[y]]], y], p2 -> FUNCTION[y],
              p3 -> subclass[composite[x, inverse[y]], composite[y, inverse[y]]],
              p4 -> FUNCTION[composite[Id, y]],
                   p5 -> subclass[composite[x, inverse[y]], Id]}]]

Out[37]= or[not[FUNCTION[y]], not[subclass[composite[x, id[domain[y]]], y]],
           subclass[composite[x, inverse[y]], Id]] == True

In[38]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

The sufficiency of the new condition now follows:

```
In[39]:= Map[not, SubstTest[and, implies[and[p2, p3], p4],
           implies[and[p1, p2, p4], p5], not[implies[and[p1, p2, p3], p5]],
             {p1 -> FUNCTION[x], p2 -> FUNCTION[y],
              p3 -> subclass[composite[x, id[domain[y]]], y],
                   p4 -> subclass[composite[x, inverse[y]], Id], p5 -> FUNCTION[union[x, y]]}]]

Out[39]= or[FUNCTION[union[x, y]], not[FUNCTION[x]],
           not[FUNCTION[y]], not[subclass[composite[x, id[domain[y]]], y]]] == True
```

```
In[40]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

some variable−free rules

In this section a variable−free version of the results of the previous section is explored. First a technical lemma:

```
In[41]:= image[inverse[CUP], FUNS] // RelnNormality // Reverse

Out[41]= composite[id[FUNS], LB[LB[complement[cross[Id, Di]]]], id[FUNS]] ==
          image[inverse[CUP], FUNS]

In[42]:= composite[id[FUNS], LB[LB[complement[cross[Id, Di]]]], id[FUNS]] :=
          image[inverse[CUP], FUNS]
```

The above rule is needed to obtain this result:

```
In[43]:= class[pair[x, y], FUNCTION[union[x, y]]]

Out[43]= image[inverse[CUP], FUNS]
```

The following result is probably too complicated to be worth making permanent:

```
In[44]:= simplify = True; cond = True;

In[45]:= Map[equal[0, composite[Id, complement[#]]] &,
           SubstTest[class, pair[x, y], implies[member[union[x, y], z],
             subclass[composite[x, id[domain[y]]], y]], z -> FUNS]] // Reverse

Out[45]= equal[0, intersection[FUNS,
            image[CUP, composite[intersection[composite[complement[S], COMPOSE], composite[
                inverse[IMAGE[FIRST]], inverse[IMAGE[DUP]], SECOND]], inverse[FIRST]]]]] == True
```

the case of disjoint domains

The union of functions with disjoint domains is important enough to warrant a separate rewrite rule.

```
In[46]:= SubstTest[implies, equal[z, w], equal[fix[z], fix[w]],
           {z -> 0, w -> composite[x, inverse[y], Di]}]

Out[46]= or[not[equal[0, intersection[domain[x], domain[y]]]],
           subclass[composite[x, inverse[y]], Id]] == True

In[47]:= or[not[equal[0, intersection[domain[x_], domain[y_]]]],
           subclass[composite[x_, inverse[y_]], Id]] := True

In[48]:= Map[not, SubstTest[and, implies[p1, p4],
           implies[and[p2, p3, p4], p5], not[implies[and[p1, p2, p3], p5]],
             {p1 -> disjoint[domain[x], domain[y]], p2 -> FUNCTION[x], p3 -> FUNCTION[y],
            p4 -> subclass[composite[x, inverse[y]], Id], p5 -> FUNCTION[union[x, y]]}]]

Out[48]= or[FUNCTION[union[x, y]], not[equal[0, intersection[domain[x], domain[y]]]],
           not[FUNCTION[x]], not[FUNCTION[y]]] == True

In[49]:= or[FUNCTION[union[x_, y_]], not[equal[0, intersection[domain[x_], domain[y_]]]],
           not[FUNCTION[x_]], not[FUNCTION[y_]]] := True
```

There is a variable−free version of this, but it is probably not worth saving:

```
In[50]:= Map[equal[0, composite[complement[#]]] &, SubstTest[class, pair[x, y],
           implies[and[member[x, z], member[y, z], disjoint[domain[x], domain[y]]],
            member[union[x, y], z]], z -> FUNS]] // Reverse

Out[50]= subclass[image[CUP, composite[id[FUNS],
           inverse[IMAGE[FIRST]], DISJOINT, IMAGE[FIRST], id[FUNS]]], FUNS] == True

In[51]:= subclass[image[CUP, composite[id[FUNS],
           inverse[IMAGE[FIRST]], DISJOINT, IMAGE[FIRST], id[FUNS]]], FUNS] := True
```

Restatement:

```
In[52]:= subclass[composite[id[FUNS], inverse[IMAGE[FIRST]], DISJOINT, IMAGE[FIRST], id[FUNS]],
          image[inverse[CUP], FUNS]]

Out[52]= True
```

lemma 1

The rest of the notebook is devoted to rederiving Theorem **FS−CUP**. The first lemma needed is this:

```
In[53]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u -> cart[x, x], v -> image[inverse[CUP], FUNS],
                 w -> composite[inverse[E], inverse[DUP]]}]

Out[53]= or[not[subclass[image[CUP, cart[x, x]], FUNS]], subclass[U[x], cart[V, V]]] == True

In[54]:= or[not[subclass[image[CUP, cart[x_, x_]], FUNS]], subclass[U[x_], cart[V, V]]] := True
```

lemma 2

```
In[55]:= SubstTest[implies, and[member[u, v], subclass[v, w]], member[u, w],
             {u -> pair[y, z], v -> cart[x, x], w -> image[inverse[CUP], FUNS]}] // MapNotNot

Out[55]= or[FUNCTION[union[y, z]], not[member[y, x]],
            not[member[z, x]], not[subclass[image[CUP, cart[x, x]], FUNS]]] == True

In[56]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed
```

Restatement:

```
In[57]:= implies[and[member[y, x], member[z, x], subclass[image[CUP, cart[x, x]], FUNS]],
          FUNCTION[union[y, z]]]

Out[57]= True
```

lemma 3

```
In[58]:= SubstTest[implies,
          and[member[pair[v, u], composite[Id, y]], member[pair[u, w], composite[Id, x]]],
          member[pair[v, w], composite[x, y]], y -> inverse[x]]

Out[58]= or[member[pair[v, w], composite[x, inverse[x]]], not[member[u, V]], not[member[v, V]],
           not[member[w, V]], not[member[pair[u, v], x]], not[member[pair[u, w], x]]] == True

In[59]:= (% /. {u -> u_, v -> v_, w -> w_, x -> x_}) /. Equal -> SetDelayed

In[60]:= SubstTest[implies, and[member[u, y], subclass[y, z]], member[u, z],
             {u -> pair[v, w], y -> composite[x, inverse[x]], z -> Id}] // MapNotNot

Out[60]= or[equal[v, w], not[FUNCTION[composite[Id, x]]],
           not[member[pair[v, w], composite[x, inverse[x]]]]] == True

In[61]:= (% /. {v -> v_, w -> w_, x -> x_}) /. Equal -> SetDelayed

In[62]:= implies[and[FUNCTION[x], member[pair[u, v], x]],
           member[pair[u, v], cart[V, V]]] // AssertTest

Out[62]= or[and[member[u, V], member[v, V]], not[FUNCTION[x]], not[member[pair[u, v], x]]] == True

In[63]:= (% /. {u -> u_, v -> v_, x -> x_}) /. Equal -> SetDelayed

In[64]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[and[p1, p7], p5], implies[and[p1, p8], p6],
                 implies[and[p2, p3], p4], implies[and[p5, p6], p3],
           not[implies[and[p1, p7, p8], p4]],
             {p1 -> FUNCTION[x], p2 -> FUNCTION[composite[Id, x]],
            p3 -> member[pair[v, w], composite[x, inverse[x]]], p4 -> equal[v, w],
                 p5 -> member[pair[u, v], composite[Id, x]],
            p6 -> member[pair[u, w], composite[Id, x]],
                      p7 -> member[pair[u, v], x], p8 -> member[pair[u, w], x]}]]

Out[64]= or[equal[v, w], not[FUNCTION[x]],
           not[member[pair[u, v], x]], not[member[pair[u, w], x]]] == True

In[65]:= (% /. {u -> u_, v -> v_, w -> w_, x -> x_}) /. Equal -> SetDelayed

In[66]:= SubstTest[implies, and[FUNCTION[z], member[pair[u, v], z], member[pair[u, w], z]],
          equal[v, w], z -> union[x, y]]

Out[66]= or[and[not[member[pair[u, v], x]], not[member[pair[u, v], y]]],
           and[not[member[pair[u, w], x]], not[member[pair[u, w], y]]],
           equal[v, w], not[FUNCTION[union[x, y]]]] == True

In[67]:= (% /. {u -> u_, v -> v_, w -> w_, x -> x_, y -> y_}) /. Equal -> SetDelayed

In[68]:= Map[not, SubstTest[and, implies[p1, p3], implies[p2, p4],
           implies[and[p3, p4, p5], p6], not[implies[and[p1, p2, p5], p6]],
             {p1 -> member[pair[u, v], x], p2 -> member[pair[u, w], y],
            p3 -> member[pair[u, v], union[x, y]],
                 p4 -> member[pair[u, w], union[x, y]],
            p5 -> FUNCTION[union[x, y]], p6 -> equal[v, w]}]]

Out[68]= or[equal[v, w], not[FUNCTION[union[x, y]]],
           not[member[pair[u, v], x]], not[member[pair[u, w], y]]] == True

In[69]:= (% /. {u -> u_, v -> v_, w -> w_, x -> x_, y -> y_}) /. Equal -> SetDelayed
```

Restatement:

```
In[70]:= implies[and[member[pair[u, v], x],
           member[pair[u, w], y], FUNCTION[union[x, y]]], equal[v, w]]

Out[70]= True
```

derivation of Theorem FS−CUP

Lemma 4:

```
In[71]:= Map[not, SubstTest[and, implies[and[p1, p2, p3], p4],
           implies[and[p4, p5, p6], p7], not[implies[and[p1, p2, p3, p5, p6], p7]],
             {p1 -> member[y, x], p2 -> member[z, x],
              p3 -> subclass[image[CUP, cart[x, x]], FUNS], p4 -> FUNCTION[union[y, z]],
                   p5 -> member[pair[u, v], y], p6 -> member[pair[u, w], z], p7 -> equal[v, w]}]]

Out[71]= or[equal[v, w], not[member[y, x]], not[member[z, x]], not[member[pair[u, v], y]],
           not[member[pair[u, w], z]], not[subclass[image[CUP, cart[x, x]], FUNS]]] == True
```

All the variables, except **x**, can be eliminated all at once (this takes a while):

```
In[72]:= Map[assert[forall[u, v, w, y, z, #]] &, %]

Out[72]= or[FUNCTION[composite[Id, U[x]]], not[subclass[image[CUP, cart[x, x]], FUNS]]] == True

In[73]:= or[FUNCTION[composite[Id, U[x_]]],
           not[subclass[image[CUP, cart[x_, x_]], FUNS]]] := True
```

The result can be cleaned up:

```
In[74]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p1, p3], implies[and[p2, p3], p4], not[implies[p1, p4]],
             {p1 -> subclass[image[CUP, cart[x, x]], FUNS],
              p2 -> subclass[U[x], cart[V, V]], p3 -> FUNCTION[composite[Id, U[x]]],
         p4 -> FUNCTION[U[x]]}]]

Out[74]= or[FUNCTION[U[x]], not[subclass[image[CUP, cart[x, x]], FUNS]]] == True
```

This is Theorem **FS−CUP** which was proved using **Otter**.

```
In[75]:= or[FUNCTION[U[x_]], not[subclass[image[CUP, cart[x_, x_]], FUNS]]] := True
```

the converse of Theorem FS−CUP

```
In[76]:= implies[and[member[u, x], member[v, x]], subclass[union[u, v], U[x]]] // NotNotTest

Out[76]= or[and[subclass[u, U[x]], subclass[v, U[x]]],
           not[member[u, x]], not[member[v, x]]] == True

In[77]:= (% /. {u -> u_, v -> v_, x -> x_}) /. Equal -> SetDelayed
```

```
In[78]:= SubstTest[implies, and[subclass[w, z], FUNCTION[z]],
          FUNCTION[w], {w -> union[u, v], z -> U[x]}]

Out[78]= or[FUNCTION[union[u, v]], not[FUNCTION[U[x]]],
           not[subclass[u, U[x]]], not[subclass[v, U[x]]]] == True

In[79]:= (% /. {u -> u_, v -> v_, x -> x_}) /. Equal -> SetDelayed

In[80]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[and[p3, p4], p5], not[implies[and[p1, p2, p4], p5]],
             {p1 -> member[u, x], p2 -> member[v, x], p3 -> subclass[union[u, v], U[x]],
            p4 -> FUNCTION[U[x]], p5 -> FUNCTION[union[u, v]]}]]

Out[80]= or[FUNCTION[union[u, v]], not[FUNCTION[U[x]]],
           not[member[u, x]], not[member[v, x]]] == True

In[81]:= (% /. {u -> u_, v -> v_, x -> x_}) /. Equal -> SetDelayed
```

The **FUNCTION** predicate can be sequestered using the following trick:

```
In[82]:= implies[and[member[pair[u, v], cart[x, x]], subclass[P[U[x]], FUNS]],
          subclass[P[union[u, v]], FUNS]]

Out[82]= True

In[83]:= Map[equal[0, composite[Id, complement[#]]] &, SubstTest[class, pair[u, v],
            implies[and[member[pair[u, v], cart[x, x]], subclass[P[U[x]], z]],
             subclass[P[union[u, v]], z]], z -> FUNS]] // Reverse

Out[83]= or[not[FUNCTION[U[x]]], subclass[image[CUP, cart[x, x]], FUNS]] == True

In[84]:= or[not[FUNCTION[U[x_]]], subclass[image[CUP, cart[x_, x_]], FUNS]] := True
```

The theorem and its converse can be combined to yield a simple rewrite rule:

```
In[85]:= equiv[subclass[image[CUP, cart[x, x]], FUNS], FUNCTION[U[x]]]

Out[85]= True
```

The orientation of the new rule is tentative:

```
In[86]:= subclass[image[CUP, cart[x_, x_]], FUNS] := FUNCTION[U[x]]
```

a variable−free corollary

Lemma:

```
In[87]:= image[inverse[BIGCUP], intersection[x, P[y]]] // Normality // Reverse

Out[87]= intersection[image[inverse[BIGCUP], x], P[P[y]]] ==
          image[inverse[BIGCUP], intersection[x, P[y]]]

In[88]:= intersection[image[inverse[BIGCUP], x_], P[P[y_]]] :=
          image[inverse[BIGCUP], intersection[x, P[y]]]
```

The following is a corollary of the result derived in the preceding section:

In[89]:= **cliques[image[inverse[CUP], FUNS]] // Normality**

Out[89]= cliques[image[inverse[CUP], FUNS]] == image[inverse[BIGCUP], FUNS]

In[90]:= **cliques[image[inverse[CUP], FUNS]] := image[inverse[BIGCUP], FUNS]**