acyclic digraphs

Johan G. F. Belinfante
2003 April 1; revised 2003 October 15

```
In[1]:= << goedel52.t01; << tools.m

        :Package Title: goedel52.t01      2003 October 9 at 5:30 a.m.

        It is now:  2003 Oct 15 at 8:26

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 September 29

        weightlimit = 40
```

introduction

A digraph or directed graph is interpreted here as a relation. It is acyclic if there are no directed paths that lead from a vertex back to the same vertex. Under this interpretation, the class **ACYCLIC** of acyclic digraphs can be characterized by the following membership condition:

```
In[2]:= member[x_, ACYCLIC] := and[member[x, V], subclass[x, cart[V, V]], equal[0, fix[trv[x]]]]
```

In the literature one often encounters a different interpretation of path that ignores direction. If one wants that interpretation, one need only replace **x** with **union[x, inverse[x]]**.

hereditary property of ACYCLIC

The transitive property of **subclass** yields:

```
In[3]:= SubstTest[implies, and[subclass[x, y], subclass[y, z]],
          subclass[x, z], z -> complement[Id]]

Out[3]= or[equal[0, fix[x]], not[equal[0, fix[y]]], not[subclass[x, y]]] == True
```

```
In[4]:= or[equal[0, fix[x_]], not[equal[0, fix[y_]]], not[subclass[x_, y_]]] := True
```

From this one derives:

```
In[5]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
            {p1 -> subclass[x, y], p2 -> subclass[trv[x], trv[y]],
           p3 -> implies[equal[0, fix[trv[y]]], equal[0, fix[trv[x]]]]}]]

Out[5]= or[equal[0, fix[trv[x]]], not[equal[0, fix[trv[y]]]], not[subclass[x, y]]] == True
```

```
In[6]:= or[equal[0, fix[trv[x_]]], not[equal[0, fix[trv[y_]]]], not[subclass[x_, y_]]] := True
```

From this one can derive the hereditary property of the class **ACYCLIC**, using the observation that **fix[trv[x]]** can be written as an image of **P[x]**.

```
In[7]:= image[composite[inverse[E], IMAGE[inverse[DUP]], HULL[TRV]], P[x]]

Out[7]= fix[trv[x]]
```

The following lemma is helpful here:

```
In[8]:= Assoc[inverse[DUP], complement[inverse[E]], composite[id[TRV], S]] // Reverse

Out[8]= composite[complement[inverse[E]], IMAGE[inverse[DUP]], id[TRV], S] ==
         composite[complement[inverse[E]], IMAGE[inverse[DUP]], HULL[TRV]]

In[9]:= composite[complement[inverse[E]], IMAGE[inverse[DUP]], id[TRV], S] :=
         composite[complement[inverse[E]], IMAGE[inverse[DUP]], HULL[TRV]]
```

With this key lemma installed, the desired result is immediate:

```
In[10]:= Map[equal[0, #] &, intersection[image[S, complement[ACYCLIC]], ACYCLIC] // Normality]

Out[10]= subclass[image[inverse[S], ACYCLIC], ACYCLIC] == True

In[11]:= subclass[image[inverse[S], ACYCLIC], ACYCLIC] := True
```

Since the reverse inclusion also holds, one obtains an equation:

```
In[12]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> image[inverse[S], ACYCLIC], v -> ACYCLIC}]

Out[12]= True == equal[ACYCLIC, image[inverse[S], ACYCLIC]]
```

This rewrite rule just says that a subgraph of an acyclic graph is acyclic.

```
In[13]:= image[inverse[S], ACYCLIC] := ACYCLIC
```

The following is another formulation of this hereditary property:

```
In[14]:= SubstTest[image, S, complement[image[inverse[S], x]], x -> ACYCLIC]

Out[14]= image[S, complement[ACYCLIC]] == complement[ACYCLIC]

In[15]:= image[S, complement[ACYCLIC]] := complement[ACYCLIC]
```

two corollaries

The hereditary property has these immediate corollaries:

```
In[16]:= SubstTest[Aclosure, image[inverse[S], x], x -> ACYCLIC]

Out[16]= Aclosure[ACYCLIC] == ACYCLIC

In[17]:= Aclosure[ACYCLIC] := ACYCLIC
```

```
In[18]:= SubstTest[HULL, image[inverse[S], x], x -> ACYCLIC]

Out[18]= HULL[ACYCLIC] == id[ACYCLIC]

In[19]:= HULL[ACYCLIC] := id[ACYCLIC]
```

As a corollary, one finds a formula for acyclic hulls:

```
In[20]:= Map[A, SubstTest[image, HULL[z], singleton[x], z -> ACYCLIC]] // Reverse

Out[20]= A[intersection[ACYCLIC, image[S, singleton[x]]]] ==
          union[x, complement[image[V, intersection[ACYCLIC, singleton[x]]]]]

In[21]:= A[intersection[ACYCLIC, image[S, singleton[x_]]]] :=
          union[x, complement[image[V, intersection[ACYCLIC, singleton[x]]]]]
```

some simplifications

Some simplification rules are needed to proceed:

```
In[22]:= SubstTest[implies, subclass[u, v],
          subclass[image[inverse[u], w], image[inverse[v], w]],
             {u -> composite[HULL[TRV], S], v -> composite[S, HULL[TRV]], w -> P[x]}]

Out[22]= subclass[image[inverse[S], image[inverse[HULL[TRV]], P[x]]],
           image[inverse[HULL[TRV]], P[x]]] == True

In[23]:= subclass[image[inverse[S], image[inverse[HULL[TRV]], P[x_]]],
           image[inverse[HULL[TRV]], P[x_]]] := True
```

The reverse inclusion also holds, yielding an equation:

```
In[24]:= SubstTest[and, subclass[u, v], subclass[v, u],
             {u -> image[inverse[S], image[inverse[HULL[TRV]], P[x]]],
           v -> image[inverse[HULL[TRV]], P[x]]}]

Out[24]= True == equal[image[inverse[S], image[inverse[HULL[TRV]], P[x]]],
           image[inverse[HULL[TRV]], P[x]]]

In[25]:= image[inverse[S], image[inverse[HULL[TRV]], P[x_]]] := image[inverse[HULL[TRV]], P[x]]
```

This hereditary property also implies:

```
In[26]:= SubstTest[image, S, complement[image[inverse[S], y]],
          y -> image[inverse[HULL[TRV]], P[x]]]

Out[26]= image[S, complement[image[inverse[HULL[TRV]], P[x]]]] ==
          complement[image[inverse[HULL[TRV]], P[x]]]

In[27]:= image[S, complement[image[inverse[HULL[TRV]], P[x_]]]] :=
          complement[image[inverse[HULL[TRV]], P[x]]]
```

The test **IminComp** can be used to derive this simplification rule:

```
In[28]:= IminComp[HULL[TRV], id[P[cart[V, V]]], x] // Reverse

Out[28]= intersection[image[inverse[HULL[TRV]], x], P[cart[V, V]]] ==
          image[inverse[HULL[TRV]], x]
```

```
In[29]:= intersection[image[inverse[HULL[TRV]], x_], P[cart[V, V]]] :=
          image[inverse[HULL[TRV]], x]
```

Another obvservation:

```
In[30]:= Assoc[HULL[TRV], id[P[cart[V, V]]], inverse[IMAGE[id[cart[V, V]]]]] // Reverse

Out[30]= composite[HULL[TRV], inverse[IMAGE[id[cart[V, V]]]]] == HULL[TRV]

In[31]:= composite[HULL[TRV], inverse[IMAGE[id[cart[V, V]]]]] := HULL[TRV]

In[32]:= IminComp[HULL[TRV], inverse[IMAGE[id[cart[V, V]]]], x] // Reverse

Out[32]= image[IMAGE[id[cart[V, V]]], image[inverse[HULL[TRV]], x]] ==
          image[inverse[HULL[TRV]], x]

In[33]:= image[IMAGE[id[cart[V, V]]], image[inverse[HULL[TRV]], x_]] :=
          image[inverse[HULL[TRV]], x]
```

a key formula

The following lemma is crucial:

```
In[34]:= ImageComp[S, id[P[y]], x] // Reverse

Out[34]= image[S, intersection[x, P[y]]] == image[inverse[IMAGE[id[y]]], image[S, x]]

In[35]:= image[S, intersection[x_, P[y_]]] := image[inverse[IMAGE[id[y]]], image[S, x]]
```

A key formula now emerges:

```
In[36]:= Map[equal[0, #] &, symdif[ACYCLIC, image[inverse[HULL[TRV]], P[Di]]] // Normality]

Out[36]= equal[ACYCLIC, image[inverse[HULL[TRV]], P[Di]]] == True

In[37]:= image[inverse[HULL[TRV]], P[Di]] := ACYCLIC
```

The following similar result will also be needed later.

```
In[38]:= IminComp[id[P[cart[V, V]]], HULL[TRV], P[union[Di, complement[cart[V, V]]]]]

Out[38]= image[inverse[HULL[TRV]], P[union[Di, complement[cart[V, V]]]]] == ACYCLIC

In[39]:= image[inverse[HULL[TRV]], P[union[Di, complement[cart[V, V]]]]] := ACYCLIC
```

transitive closure results

The key fact derived in the preceding section has many corollaries; for example:

```
In[40]:= IminComp[HULL[TRV], HULL[TRV], P[Di]] // Reverse

Out[40]= image[inverse[HULL[TRV]], ACYCLIC] == ACYCLIC
```

```
In[41]:= image[inverse[HULL[TRV]], ACYCLIC] := ACYCLIC

In[42]:= ImageComp[HULL[TRV], inverse[HULL[TRV]], P[Di]] // Reverse

Out[42]= image[HULL[TRV], ACYCLIC] == intersection[TRV, P[Di]]

In[43]:= image[HULL[TRV], ACYCLIC] := intersection[TRV, P[Di]]
```

Similarly,

```
In[44]:= ImageComp[HULL[TRV], inverse[HULL[TRV]], ACYCLIC]

Out[44]= intersection[ACYCLIC, TRV] == intersection[TRV, P[Di]]

In[45]:= intersection[ACYCLIC, TRV] := intersection[TRV, P[Di]]
```

Corollary:

```
In[46]:= SubstTest[implies, subclass[u, v], subclass[image[u, w], image[v, w]],
             {u -> id[TRV], v -> inverse[HULL[TRV]], w -> P[Di]}]

Out[46]= subclass[intersection[TRV, P[Di]], ACYCLIC] == True

In[47]:= subclass[intersection[TRV, P[Di]], ACYCLIC] := True
```

a sequestering technique

The trick used to sequester **WELLFOUNDED** in the derivation of the theorem about well−founded induction can be applied to any hereditary predicate. Here this technique is applied to the case of acyclic relations. The statement **subclass[P[x], ACYCLIC]** is shown to be logically equivalent to the statement **and[subclass[x, cart[V,V]], equal[0, fix[trv[x]]]]**. The main work is done in a single step:

```
In[48]:= SubstTest[equal, 0, image[z, P[x]],
           z -> composite[inverse[DUP], inverse[E], HULL[TRV]]] // Reverse

Out[48]= subclass[P[composite[Id, x]], ACYCLIC] == equal[0, fix[trv[x]]]

In[49]:= subclass[P[composite[Id, x_]], ACYCLIC] := equal[0, fix[trv[x]]]
```

The rest is done using **AssertTest**:

```
In[50]:= subclass[P[x], ACYCLIC] // AssertTest

Out[50]= subclass[P[x], ACYCLIC] == and[equal[0, fix[trv[x]]], subclass[x, cart[V, V]]]

In[51]:= subclass[P[x_], ACYCLIC] := and[equal[0, fix[trv[x]]], subclass[x, cart[V, V]]]
```

a formula for U[ACYCLIC]

```
In[52]:= Map[equal[0, #] &, dif[ACYCLIC, P[cart[V, V]]] // Normality]

Out[52]= subclass[U[ACYCLIC], cart[V, V]] == True
```

```
In[53]:= subclass[U[ACYCLIC], cart[V, V]] := True
```

This implies an obvious fact:

```
In[54]:= equal[intersection[ACYCLIC, P[cart[V, V]]], ACYCLIC]

Out[54]= True
```

The corresponding rewrite rule is added:

```
In[55]:= intersection[ACYCLIC, P[cart[V, V]]] := ACYCLIC
```

Here is a related result:

```
In[56]:= ImageComp[IMAGE[id[cart[V, V]]], id[P[cart[V, V]]], ACYCLIC] // Reverse

Out[56]= image[IMAGE[id[cart[V, V]]], ACYCLIC] == ACYCLIC

In[57]:= image[IMAGE[id[cart[V, V]]], ACYCLIC] := ACYCLIC
```

Another related result:

```
In[58]:= IminComp[IMAGE[id[cart[V, V]]], S, ACYCLIC] // Reverse

Out[58]= image[inverse[S], image[inverse[IMAGE[id[cart[V, V]]]], ACYCLIC]] ==
          image[inverse[IMAGE[id[cart[V, V]]]], ACYCLIC]

In[59]:= image[inverse[S], image[inverse[IMAGE[id[cart[V, V]]]], ACYCLIC]] :=
          image[inverse[IMAGE[id[cart[V, V]]]], ACYCLIC]
```

Better results can be derived.

```
In[60]:= SubstTest[implies, subclass[u, v], subclass[image[u, w], image[v, w]],
             {u -> inverse[HULL[TRV]], v -> inverse[S], w -> P[Di]}]

Out[60]= equal[0, fix[U[ACYCLIC]]] == True
```

This temporary rule will soon be replaced with a better one. An upper bound for **ACYCLIC** is derived as follows:

```
In[61]:= fix[U[ACYCLIC]] := 0
```

Restatement:

```
In[62]:= subclass[ACYCLIC, P[Di]]

Out[62]= True
```

To derive a lower bound for **ACYCLIC**, we use some examples. A cartesian product of two disjoint sets is an acyclic relation.

```
In[63]:= IminComp[composite[IMAGE[inverse[DUP]], HULL[TRV]], CART, singleton[0]] // Reverse

Out[63]= image[inverse[CART], ACYCLIC] == DISJOINT

In[64]:= image[inverse[CART], ACYCLIC] := DISJOINT
```

Specializing to singletons, one finds:

```
In[65]:= IminComp[CART, cross[SINGLETON, SINGLETON], ACYCLIC]

Out[65]= composite[Id, image[inverse[SINGLETON], ACYCLIC]] == Di

In[66]:= composite[Id, image[inverse[SINGLETON], ACYCLIC]] := Di
```

This yields a useful lower bound:

```
In[67]:= SubstTest[subclass, composite[Id, z], z, z -> image[inverse[SINGLETON], ACYCLIC]]

Out[67]= subclass[image[SINGLETON, Di], ACYCLIC] == True

In[68]:= subclass[image[SINGLETON, Di], ACYCLIC] := True
```

The desired result is obtained by taking the union of all these singleton examples.

```
In[69]:= SubstTest[implies, subclass[u, v], subclass[U[u], U[v]],
             {u -> image[SINGLETON, Di], v -> ACYCLIC}]

Out[69]= subclass[Di, U[ACYCLIC]] == True

In[70]:= subclass[Di, U[ACYCLIC]] := True

In[71]:= SubstTest[and, subclass[u, v], subclass[v, u],
             {u -> U[ACYCLIC], v -> Di}]

Out[71]= True == equal[Di, U[ACYCLIC]]
```

This yields the following important formula:

```
In[72]:= U[ACYCLIC] := Di
```

ACYCLIC is a proper class

The fact that **ACYCLIC** is a proper class follows from the formula for **U[ACYCLIC]** derived in the preceding section.

```
In[73]:= SubstTest[member, U[x], V, x -> ACYCLIC] // Reverse

Out[73]= member[ACYCLIC, V] == False

In[74]:= member[ACYCLIC, V] := False
```

More generally:

```
In[75]:= member[ACYCLIC, x] // AssertTest

Out[75]= member[ACYCLIC, x] == False

In[76]:= member[ACYCLIC, x_] := False
```

Uclosure

The following corollaries follow from the fact that **ACYCLIC** is hereditary plus the formula for **U[ACYCLIC]** derived in the preceding section.

```
In[77]:= SubstTest[Uclosure, image[inverse[S], x], x -> ACYCLIC]

Out[77]= Uclosure[ACYCLIC] == P[Di]

In[78]:= Uclosure[ACYCLIC] := P[Di]

In[79]:= SubstTest[CORE, Uclosure[x], x -> ACYCLIC] // Reverse

Out[79]= CORE[ACYCLIC] == IMAGE[id[Di]]

In[80]:= CORE[ACYCLIC] := IMAGE[id[Di]]
```

A corresponding result can be obtained for the acyclic core **U[intersection[ACYCLIC, P[x]]]** without assuming that **x** is a set.

```
In[81]:= ImageComp[inverse[E], composite[id[ACYCLIC], inverse[S]], P[x]] // Reverse

Out[81]= U[intersection[ACYCLIC, P[x]]] == intersection[Di, x]

In[82]:= U[intersection[ACYCLIC, P[x_]]] := intersection[Di, x]
```

an alternate membership rule

The following are two equivalent descriptions of **ACYCLIC**.

```
In[83]:= class[x, equal[0, fix[A[intersection[TRV, image[S, singleton[x]]]]]]]

Out[83]= ACYCLIC

In[84]:= class[x, and[subclass[x, cart[V, V]], equal[0, fix[trv[x]]]]]

Out[84]= ACYCLIC
```

This implies an alternate membership rule for **ACYCLIC** which one can derive using the following fact:

```
In[85]:= equal[A[intersection[TRV, image[S, singleton[x]]]],
          union[trv[x], complement[image[V, intersection[singleton[x], P[cart[V, V]]]]]]]

Out[85]= True
```

Lemma.

```
In[86]:= implies[equal[u, v], equiv[equal[u, 0], equal[v, 0]]] // NotNotTest

Out[86]= or[and[equal[0, u], equal[0, v]],
           and[not[equal[0, u]], not[equal[0, v]]], not[equal[u, v]]] == True
```

```
In[87]:= (% /. {u -> u_, v -> v_}) /. Equal -> SetDelayed

In[88]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 -> equal[u, v], p2 -> equal[fix[u], fix[v]],
         p3 -> equiv[equal[0, fix[u]], equal[0, fix[v]]]}]]

Out[88]= or[and[equal[0, fix[u]], equal[0, fix[v]]],
          and[not[equal[0, fix[u]]], not[equal[0, fix[v]]]], not[equal[u, v]]] == True

In[89]:= (% /. {u -> u_, v -> v_}) /. Equal -> SetDelayed

In[90]:= SubstTest[implies, equal[u, v], equiv[equal[0, fix[u]], equal[0, fix[v]]],
            {u -> A[intersection[TRV, image[S, singleton[x]]]],
          v -> union[trv[x], complement[image[V, intersection[singleton[x], P[cart[V, V]]]]]]}]

Out[90]= or[and[not[equal[0, fix[A[intersection[TRV, image[S, singleton[x]]]]]]],
           not[equal[0, fix[trv[x]]]],
          and[not[equal[0, fix[A[intersection[TRV, image[S, singleton[x]]]]]],
           not[member[x, V]]],
          and[not[equal[0, fix[A[intersection[TRV, image[S, singleton[x]]]]]],
           not[subclass[x, cart[V, V]]]],
          and[equal[0, fix[A[intersection[TRV, image[S, singleton[x]]]]]],
           equal[0, fix[trv[x]]], member[x, V], subclass[x, cart[V, V]]]] == True

In[91]:= (% /. x -> x_) /. Equal -> SetDelayed

In[92]:= equiv[equal[0, fix[A[intersection[TRV, image[S, singleton[x]]]]]],
          and[equal[0, fix[trv[x]]], member[x, V], subclass[x, cart[V, V]]]]

Out[92]= True

In[93]:= equal[0, fix[A[intersection[TRV, image[S, singleton[x_]]]]]] :=
          and[equal[0, fix[trv[x]]], member[x, V], subclass[x, cart[V, V]]]
```

This provides an alternate membership rule that could be used to define the class **ACYCLIC**.

```
In[94]:= equiv[member[x, ACYCLIC], equal[0, fix[A[intersection[TRV, image[S, singleton[x]]]]]]]

Out[94]= True
```

The alternate membership rule was used in an earlier version of this notebook (dated 2003 April 1), but was abandoned because it made deriving facts about the class of acyclic relations more difficult.

inverses of acyclic relations

The sequestering trick can be used to obtain an easy derivation of the fact that inverses of acyclic relations are acyclic:

```
In[95]:= SubstTest[class, x,
           and[subclass[x, cart[V, V]], subclass[P[inverse[x]], z]], z -> ACYCLIC] // Reverse

Out[95]= image[INVERSE, ACYCLIC] == ACYCLIC

In[96]:= image[INVERSE, ACYCLIC] := ACYCLIC
```

Here are some related results:

```
In[97]:= ImageComp[IMAGE[SWAP], id[P[cart[V, V]]], ACYCLIC] // Reverse

Out[97]= image[IMAGE[SWAP], ACYCLIC] == ACYCLIC

In[98]:= image[IMAGE[SWAP], ACYCLIC] := ACYCLIC

In[99]:= SubstTest[class, x, subclass[P[inverse[x]], z], z -> ACYCLIC] // Reverse

Out[99]= image[inverse[IMAGE[SWAP]], ACYCLIC] == image[inverse[IMAGE[id[cart[V, V]]]], ACYCLIC]

In[100]:=
         image[inverse[IMAGE[SWAP]], ACYCLIC] := image[inverse[IMAGE[id[cart[V, V]]]], ACYCLIC]
```

well−founded relations are acyclic

The theorem about transitive closures of well−founded relations implies that well−founded relations are acyclic. The proof uses the sequester trick. Since the transitive closure of a well−founded relation is well founded, and since well−founded relations are irreflexive, one finds:

```
In[101]:=
         SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
             {u -> WF, v -> P[Di], w -> inverse[HULL[TRV]]}]

Out[101]=
         subclass[WF, ACYCLIC] == True

In[102]:=
         subclass[WF, ACYCLIC] := True
```

The result is not limited to well−founded relations that are sets. The results of the preceding section can be used to generalize this result to arbitrary well−founded relations:

```
In[103]:=
         SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
             {u -> P[x], v -> WF, w -> ACYCLIC}] // MapNotNot

Out[103]=
         or[equal[0, fix[trv[x]]], not[WELLFOUNDED[x]]] == True

In[104]:=
         or[equal[0, fix[trv[x_]]], not[WELLFOUNDED[x_]]] := True
```

rules not used

This section contains simplification rules discovered in the course of this work that turned out not to be needed. The first two are general rules that can be used to eliminate **IMAGE[id[x]]** in certain expressions involving complements:

```
In[105]:=
         union[complement[image[inverse[IMAGE[id[x]]], y]], complement[P[x]]] // DoubleComplement

Out[105]=
         union[complement[image[inverse[IMAGE[id[x]]], y]], complement[P[x]]] ==
          union[complement[y], complement[P[x]]]
```

```
In[106]:=
union[complement[image[inverse[IMAGE[id[x_]]], y_]], complement[P[x_]]] :=
 union[complement[y], complement[P[x]]]

In[107]:=
equal[intersection[complement[image[inverse[IMAGE[id[x]]], y]], P[x]],
  intersection[complement[y], P[x]]] // AssertTest

Out[107]=
equal[intersection[complement[y], P[x]],
  intersection[complement[image[inverse[IMAGE[id[x]]], y]], P[x]]] == True

In[108]:=
intersection[complement[image[inverse[IMAGE[id[x_]]], y_]], P[x_]] :=
 intersection[complement[y], P[x]]
```

The other two rules are more special:

```
In[109]:=
union[complement[ACYCLIC], complement[P[cart[V, V]]]] // DoubleComplement

Out[109]=
union[complement[ACYCLIC], complement[P[cart[V, V]]]] == complement[ACYCLIC]

In[110]:=
union[complement[ACYCLIC], complement[P[cart[V, V]]]] := complement[ACYCLIC]

In[111]:=
composite[inverse[S], id[TRV],
  inverse[IMAGE[inverse[DUP]]], complement[E]] // DoubleInverse

Out[111]=
composite[inverse[S], id[TRV], inverse[IMAGE[inverse[DUP]]], complement[E]] ==
 composite[inverse[HULL[TRV]], inverse[IMAGE[inverse[DUP]]], complement[E]]

In[112]:=
composite[inverse[S], id[TRV], inverse[IMAGE[inverse[DUP]]], complement[E]] :=
 composite[inverse[HULL[TRV]], inverse[IMAGE[inverse[DUP]]], complement[E]]
```

comment

In the introduction it was noted that the term acyclic is often used to describe the more restricted class of digraphs with no cyclic semipaths, that is, digraphs with no cycles when one ignories the directions of the edges. A formula for this subclass is derived in this section.

```
In[113]:=
fix[composite[INVERSE, image[inverse[CUP], ACYCLIC]]] // Normality // Reverse

Out[113]=
intersection[fix[composite[inverse[IMAGE[SWAP]],
    image[inverse[CUP], image[inverse[IMAGE[id[cart[V, V]]], ACYCLIC]]]],
  P[cart[V, V]]] == fix[composite[INVERSE, image[inverse[CUP], ACYCLIC]]]

In[114]:=
intersection[fix[composite[inverse[IMAGE[SWAP]],
    image[inverse[CUP], image[inverse[IMAGE[id[cart[V, V]]], ACYCLIC]]]],
  P[cart[V, V]]] := fix[composite[INVERSE, image[inverse[CUP], ACYCLIC]]]
```

```
In[115]:=
class[x, member[union[x, inverse[x]], ACYCLIC]]

Out[115]=
fix[composite[INVERSE, image[inverse[CUP], ACYCLIC]]]
```

The following argument establishes the fairly obvious fact that this is in fact a subclass of **ACYCLIC**.

```
In[116]:=
subclass[composite[CUP, id[INVERSE], inverse[FIRST]], S] // AssertTest

Out[116]=
subclass[composite[CUP, id[INVERSE], inverse[FIRST]], S] == True

In[117]:=
subclass[composite[CUP, id[INVERSE], inverse[FIRST]], S] := True

In[118]:=
SubstTest[implies, subclass[u, v], subclass[image[u, w], image[v, w]],
    {u -> composite[inverse[DUP], cross[INVERSE, Id], inverse[CUP]],
  v -> inverse[S], w -> ACYCLIC}]

Out[118]=
subclass[fix[composite[INVERSE, image[inverse[CUP], ACYCLIC]]], ACYCLIC] == True

In[119]:=
subclass[fix[composite[INVERSE, image[inverse[CUP], ACYCLIC]]], ACYCLIC] := True
```