

allclosed[x]

Johan G. F. Belinfante
2004 May 4

```
In[1]:= << goedel57.04d; << tools.m

:Package Title: goedel57.04d      2004 May 4 at 1:00 p.m.

It is now: 2004 May 5 at 8:54

Loading Simplification Rules

TOOLS.M                          Revised 2004 April 26

weightlimit = 40
```

summary

It is shown that the class of all sets invariant under an infinitary operation is itself invariant under arbitrary intersections. Along the way, some other useful results are obtained. The class of all sets that are closed under an infinitary operation x is here given the temporary name **allclosed[x]**.

```
In[2]:= class[y, forall[u, v, implies[and [subclass[u, y], member[pair[u, v], x]], member[v, y]]]]
Out[2]= fix[composite[S, IMAGE[x], POWER]]

In[3]:= allclosed[x_] := fix[composite[S, IMAGE[x], POWER]]
```

The reason for not making this a permanent definition is that only two special cases currently have applications in the **GOEDEL** program.

```
In[4]:= allclosed[BIGCAP]
Out[4]= fix[ACLOSURE]

In[5]:= allclosed[BIGCUP]
Out[5]= fix[UCLOSURE]
```

These special cases are already known to be closed under arbitrary intersections, so nothing much is gained at the moment by proving that this property is general. It is being done mainly as an exercise, and for the possibility that at some future time additional applications will materialize. Note by the way that the class **invar[x]** of all sets invariant under a unary operation x is also a special case, but here again the main result is already known.

```
In[6]:= class[y, invariant[x, y]]
Out[6]= invar[x]

In[7]:= allclosed[composite[x, inverse[E]]]
Out[7]= invar[x]
```

lemmas

It is necessary to add several new rewrite rule to obtain a useful membership rule for **allclosed[x]**.

```
In[8]:= member[x, image[S, y]] // AssertTest
Out[8]= member[x, image[S, y]] == and[member[x, V], not[equal[0, intersection[y, P[x]]]]]

In[9]:= member[x_, image[S, y_]] := and[member[x, V], not[equal[0, intersection[y, P[x]]]]]

In[10]:= Map[and[member[y, z], #] &,
  SubstTest[member, y, domain[VERTSECT[w]], w -> composite[x, inverse[E]]]]
Out[10]= and[member[y, z], subclass[y, domain[VERTSECT[x]]]] ==
  and[member[y, z], member[image[x, y], V]]

In[11]:= and[member[y_, z_], subclass[y_, domain[VERTSECT[x_]]]] :=
  and[member[y, z], member[image[x, y], V]]

In[12]:= Map[and[member[y, z], #] &,
  SubstTest[and, member[w, V], subclass[w, domain[VERTSECT[x]]], w -> P[y]]]
Out[12]= and[member[y, z], subclass[P[y], domain[VERTSECT[x]]]] ==
  and[member[y, z], member[image[x, P[y]], V]]

In[13]:= and[member[y_, z_], subclass[P[y_], domain[VERTSECT[x_]]]] :=
  and[member[y, z], member[image[x, P[y]], V]]
```

These results yield the following membership rule:

```
In[14]:= member[y, allclosed[x]]
Out[14]= and[member[y, V], subclass[image[x, P[y]], y]]
```

Incidentally, we note that replacing **x** with **composite[x,inverse[S]]** leaves **allclosed** unchanged:

```
In[15]:= allclosed[composite[x, inverse[S]]] == allclosed[x]
Out[15]= True
```

derivation

Lemma.

```
In[16]:= SubstTest[implies, subclass[u, v], subclass[image[x, u], image[x, v]],
  {u -> P[y], v -> P[z]}]
Out[16]= or[not[subclass[y, z]], subclass[image[x, P[y]], image[x, P[z]]]] == True

In[17]:= or[not[subclass[y_, z_]], subclass[image[x_, P[y_]], image[x_, P[z_]]]] := True
```

Theorem.

```
In[18]:= Map[not, SubstTest[and, implies[p1, p3], implies[p3, p4], implies[and[p2, p4], p5],
  not[implies[and[p1, p2], p5]],
  {p1 -> member[y, z], p2 -> subclass[image[x, P[y]], y], p3 -> subclass[A[z], y],
  p4 -> subclass[image[x, P[A[z]]], image[x, P[y]]],
  p5 -> subclass[image[x, P[A[z]]], y]}]]
```

```
Out[18]= or[not[member[y, z]],
  not[subclass[image[x, P[y]], y], subclass[image[x, P[A[z]]], y]] == True
```

```
In[19]:= or[not[member[y_, z_]], not[subclass[image[x_, P[y_]], y_]],
  subclass[image[x_, P[A[z_]]], y_] := True
```

```
In[20]:= Map[equal[V, #] &, SubstTest[class, y,
  or[not[member[y, z]], not[subclass[image[x, P[y]], y], subclass[w, y]],
  w -> image[x, P[A[z]]]]] // Reverse
```

```
Out[20]= subclass[image[x, P[A[z]]],
  A[intersection[z, fix[composite[S, IMAGE[x], POWER]]]] == True
```

```
In[21]:= (% /. {x -> x_, z -> z_}) /. Equal -> SetDelayed
```

Lemma.

```
In[22]:= SubstTest[implies, equal[x, z], equal[A[x], A[z]], z -> intersection[x, y]]
```

```
Out[22]= or[equal[A[x], A[intersection[x, y]]], not[subclass[x, y]] == True
```

```
In[23]:= or[equal[A[x_], A[intersection[x_, y_]]], not[subclass[x_, y_]] := True
```

```
In[24]:= SubstTest[implies, and[subclass[u, v], equal[v, w]], subclass[u, w],
  {u -> image[x, P[A[z]]], v -> A[intersection[z, allclosed[x]]], w -> A[z]}]
```

```
Out[24]= or[not[equal[A[z], A[intersection[z, fix[composite[S, IMAGE[x], POWER]]]]],
  subclass[image[x, P[A[z]]], A[z]] == True
```

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

Main result:

```
In[26]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
  not[implies[p1, p3]], {p1 -> subclass[z, allclosed[x]],
  p2 -> equal[A[z], A[intersection[z, allclosed[x]]],
  p3 -> subclass[image[x, P[A[z]]], A[z]}]]]
```

```
Out[26]= or[not[subclass[z, fix[composite[S, IMAGE[x], POWER]]],
  subclass[image[x, P[A[z]]], A[z]] == True
```

```
In[27]:= or[not[subclass[z_, fix[composite[S, IMAGE[x_], POWER]]],
  subclass[image[x_, P[A[z_]]], A[z_]] := True
```

```
In[28]:= SubstTest[and, member[z, V], subclass[P[z], domain[VERTSECT[x]]], z -> A[y]]
```

```
Out[28]= and[not[equal[0, y]], subclass[P[A[y]], domain[VERTSECT[x]]] ==
  and[member[image[x, P[A[y]]], V], not[equal[0, y]]]
```

```
In[29]:= and[not[equal[0, y_]], subclass[P[A[y_]], domain[VERTSECT[x_]]] :=
  and[member[image[x, P[A[y]]], V], not[equal[0, y]]]
```

The following lemma is crucial!

```

In[30]:= SubstTest[member, y, allclosed[x], y -> A[z]]

Out[30]= and[member[image[x, P[A[z]]], V], not[equal[0, z]], subclass[image[x, P[A[z]]], A[z]]] ==
and[not[equal[0, z]], subclass[image[x, P[A[z]]], A[z]]]

In[31]:= and[member[image[x_, P[A[z_]]], V],
not[equal[0, z_]], subclass[image[x_, P[A[z_]]], A[z_]]] :=
and[not[equal[0, z]], subclass[image[x, P[A[z]]], A[z]]]

Restatement:

In[32]:= implies[subclass[z, allclosed[x]], or[equal[0, z], member[A[z], allclosed[x]]]]

Out[32]= True

In[33]:= Map[equal[V, #] &, SubstTest[class, z, implies[subclass[z, w],
or[equal[0, z], member[A[z], w]]], w -> allclosed[x]]] // Reverse

Out[33]= subclass[P[fix[composite[S, IMAGE[x], POWER]]], union[
image[inverse[BIGCAP], fix[composite[S, IMAGE[x], POWER]]], singleton[0]]] == True

In[34]:= (% /. x -> x_) /. Equal -> SetDelayed

In[35]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
{u -> P[allclosed[x]],
v -> union[image[inverse[BIGCAP], allclosed[x]], singleton[0]],
w -> BIGCAP}]

Out[35]= subclass[Aclosure[fix[composite[S, IMAGE[x], POWER]]],
fix[composite[S, IMAGE[x], POWER]]] == True

In[36]:= (% /. x -> x_) /. Equal -> SetDelayed

In[37]:= SubstTest[and, subclass[u, v], subclass[v, u],
{u -> Aclosure[allclosed[x]], v -> allclosed[x]}]

Out[37]= True == equal[Aclosure[fix[composite[S, IMAGE[x], POWER]]],
fix[composite[S, IMAGE[x], POWER]]]

In[38]:= Aclosure[fix[composite[S, IMAGE[x_], POWER]]] := fix[composite[S, IMAGE[x], POWER]]

```