Aclosure of a nest

Johan G. F. Belinfante
2006 April 20

```
In[1]:= SetDirectory["l:"]; << goedel80.16a; << tools.m

        :Package Title: goedel80.16a          2006 April 16 at 10:30 p.m.

        It is now:  2006 Apr 20 at 12:57

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 March 7

        weightlimit = 40
```

summary

A **nest** of sets is a collection of sets that is totally ordered by inclusion. The condition for a class **x** to be a nest is:

```
In[2]:= TOTALORDER[restrict[S, x, x]]

Out[2]= subclass[cart[x, x], union[S, inverse[S]]]
```

The **Uclosure** of a nest of sets is a nest. A double application of **assert** sufficed to establish this result.

```
In[3]:= subclass[cart[Uclosure[x], Uclosure[x]], union[S, inverse[S]]]

Out[3]= subclass[cart[x, x], union[S, inverse[S]]]
```

In this notebook, an analogous result is established for **Aclosure**. In the latter case, using **assert** alone does not suffice.

fix[HULL[x]] versus Aclosure[x]

In this section a brief explanation is given to help motivate a technical parametrization step encountered in a later section. The **Aclosure** of a class **x** is defined as the class of all intersections of subsets of **x**. This definition is exactly similar to the definition of the **Uclosure** of **x**, except that unions are replaced with intersections.

```
In[4]:= class[w, exists[y, and[subclass[y, x], equal[w, A[y]]]]]

Out[4]= Aclosure[x]
```

When **x** is a set, this is perfectly satsfactory, but for proper classes the **Aclosure** unfortunately does not behave quite as nicely as the **Uclosure**. For example, **Uclosure** is idempotent, but it remains unknown whether **Aclosure** is idempotent for all proper classes. (No counterexample is known.) For proper classes, one gets better results by considering intersections of

subclasses instead of intersections of subsets. Of course, Gödel's algorithm for class formation does not permit quantifiers over proper classes, so the definition requires one to a family of subclasses parametrized by a set variable:

```
In[5]:= class[w, exists[y, equal[w, A[intersection[x, image[S, set[y]]]]]]]
Out[5]= fix[HULL[x]]
```

The classes **Aclosure[x]** and **fix[HULL[x]]** are equal when **x** is a set.

```
In[6]:= implies[member[x, V], equal[Aclosure[x], fix[HULL[x]]]]
Out[6]= True
```

In principle, for proper classes there could be a difference, even though no example of this is currently known. All that has been proved to date is an inclusion:

```
In[7]:= subclass[Aclosure[x], fix[HULL[x]]]
Out[7]= True
```

From a theoretical standpoint, **fix[HULL[x]]** is better behaved than **Aclosure[x]**. For example, one can prove that **fix[HULL[x]]** is idempotent, but it is not known whether
Aclosure is idempotent:

```
In[8]:= fix[HULL[fix[HULL[x]]]]
Out[8]= fix[HULL[x]]
```

```
In[9]:= Aclosure[Aclosure[x]]
Out[9]= Aclosure[Aclosure[x]]
```

For sets, of course, there is no difference, and so **Aclosure** is idempotent in that case:

```
In[10]:= implies[member[x, V], equal[Aclosure[Aclosure[x]], Aclosure[x]]]
Out[10]= True
```

Because of this state of affairs, it is generally speaking preferable to prove theorems about **fix[HULL[x]]**, and then to deduce theorems about **Aclosure[x]** as a corollary, if desired.

comparable elements and nests

It is convenient to introduce some temporary terminology. Sets **u** and **v** are said to be **comparable** if one of them is contained in the other:

```
In[11]:= comparable[u_, v_] := or[subclass[u, v], subclass[v, u]]
```

Note the the condition for all elements of a class **x** to be comparable with all elements of a class **y** is

```
In[12]:= assert[forall[u, v, implies[and[member[u, x], member[v, y]], comparable[u, v]]]]

Out[12]= subclass[cart[x, y], union[S, inverse[S]]]
```

Lemma.

```
In[13]:= subclass[x, union[image[S, set[v]], P[v]]] // AssertTest // Reverse

Out[13]= subclass[v, A[intersection[x, complement[P[v]]]]] ==
          subclass[x, union[image[S, set[v]], P[v]]]

In[14]:= subclass[v_, A[intersection[x_, complement[P[v_]]]]] :=
          subclass[x, union[image[S, set[v]], P[v]]]
```

The condition that a set **v** be comparable with every element of a class **x** is:

```
In[15]:= assert[forall[u, implies[member[u, x], comparable[u, v]]]]

Out[15]= subclass[x, union[image[S, set[v]], P[v]]]
```

A **nest** is a class for which any pair of elements are comparable.

```
In[16]:= nest[x_] := subclass[cart[x, x], union[S, inverse[S]]]
```

The class of all nests is:

```
In[17]:= class[x, nest[x]]

Out[17]= cliques[union[S, inverse[S]]]
```

U[x] theorem

In this section it is shown that if all elements of a class **x** are comparable with a set **y**, then **U[x]** is comparable with **y**. The key fact that is used for this purpose is the following observation:

```
In[18]:= implies[subclass[u, union[v, w]], or[subclass[u, v], not[disjoint[u, w]]]]

Out[18]= True
```

From this key observation, one obtains this lemma:

```
In[19]:= SubstTest[implies, subclass[u, union[v, w]],
          or[subclass[u, v], not[disjoint[u, w]]], {u → x, v → P[y], w → image[S, set[y]]}]

Out[19]= or[member[y, image[inverse[S], x]],
           not[subclass[x, union[image[S, set[y]], P[y]]]], subclass[U[x], y]] == True

In[20]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem: If every member of **x** is comparable with **y**, then so is **U[x]**.

```
In[21]:= Map[not, SubstTest[and, implies[p1, or[p2, p3]], implies[p2, p4],
           not[implies[p1, or[p3, p4]]], {p1 -> subclass[x, union[image[S, set[y]], P[y]]], p2 ->
             member[y, image[inverse[S], x]], p3 -> subclass[U[x], y], p4 → subclass[y, U[x]]}]]

Out[21]= or[not[subclass[x, union[image[S, set[y]], P[y]]]],
           subclass[y, U[x]], subclass[U[x], y]] == True

In[22]:= or[not[subclass[x_, union[image[S, set[y_]], P[y_]]]],
           subclass[y_, U[x_]], subclass[U[x_], y_]] := True
```

In the next section, an analogous theorem is derived for intersections.

A[x] theorem

The same key observation used in the preceding section is used again, but the roles of **v** and **w** are reversed this time.

```
In[23]:= SubstTest[implies, subclass[u, union[v, w]],
          or[subclass[u, v], not[disjoint[u, w]]], {u → x, v → image[S, set[y]], w → P[y]}]

Out[23]= or[not[equal[0, intersection[x, P[y]]]],
           not[subclass[x, union[image[S, set[y]], P[y]]]], subclass[y, A[x]]] == True

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

The condition that **x** and **P[y]** are not disjoint implies that both hold some element **w** in common. Introducing this element explicitly, one derives:

```
In[25]:= Map[not, SubstTest[and, implies[p1, p3],
           implies[and[p2, p3], p4], not[implies[and[p1, p2], p4]], {p1 → member[w, x],
            p2 → subclass[w, y], p3 → subclass[A[x], w], p4 → subclass[A[x], y]}]]

Out[25]= or[not[member[w, x]], not[subclass[w, y]], subclass[A[x], y]] == True

In[26]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Eliminating the variable **w** yields:

```
In[27]:= Map[equal[V, #] &, SubstTest[class, w,
            or[not[member[w, x]], not[subclass[w, y]], subclass[z, y]], z → A[x]]] // Reverse

Out[27]= or[equal[0, intersection[x, P[y]]], subclass[A[x], y]] == True

In[28]:= or[equal[0, intersection[x_, P[y_]]], subclass[A[x_], y_]] := True
```

Combining this with our initial lemma yields the desired theorem: if every element of **x** is comparable with **y**, then **A[x]** is comparable with **y**.

```
In[29]:= Map[not, SubstTest[and, implies[p1, or[p2, p3]], implies[p2, p4],
           not[implies[p1, or[p3, p4]]], {p1 -> subclass[x, union[image[S, set[y]], P[y]]],
            p2 → not[disjoint[x, P[y]]], p3 → subclass[y, A[x]], p4 → subclass[A[x], y]}]]

Out[29]= or[not[subclass[x, union[image[S, set[y]], P[y]]]],
           subclass[y, A[x]], subclass[A[x], y]] == True

In[30]:= or[not[subclass[x_, union[image[S, set[y_]], P[y_]]]],
           subclass[A[x_], y_], subclass[y_, A[x_]]] := True
```

---

a corollary

Lemma.

```
In[31]:= SubstTest[implies, and[subclass[r, s], subclass[s, t]], subclass[r, t],
          {r → cart[u, set[v]], s → cart[x, y], t → union[S, inverse[S]]}]

Out[31]= or[not[member[v, y]], not[subclass[u, x]],
           not[subclass[cart[x, y], union[S, inverse[S]]]],
           subclass[u, union[image[S, set[v]], P[v]]], subclass[v, A[u]]] == True

In[32]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. Suppose that every element of **x** is comparable with every element of **y**. If **u** is a subclass of **x** and **v** is an element of **y**, then **A[u]** is comparable with **v**.

```
In[33]:= Map[not, SubstTest[and, implies[and[p1, p2, p3], or[p4, p5]], implies[p4, or[p5, p6]],
           not[implies[and[p1, p2, p3], or[p5, p6]]], {p1 → subclass[u, x],
            p2 → member[v, y], p3 → subclass[cart[x, y], union[S, inverse[S]]],
            p4 -> subclass[u, union[image[S, set[v]], P[v]]],
            p5 -> subclass[v, A[u]], p6 → subclass[A[u], v]}]]

Out[33]= or[not[member[v, y]], not[subclass[u, x]],
           not[subclass[cart[x, y], union[S, inverse[S]]]],
           subclass[v, A[u]], subclass[A[u], v]] == True

In[34]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

The basic idea is now to remove the variables **u** and **v**. Note that **u** need not be a set in general. To obtain the best possible result, the class **u** will be replaced with a family of classes parametrized by a set variable **w**, and then that set variable will be eliminated. The parametrization of **u** is accomplished as follows:

```
In[35]:= SubstTest[or, not[member[v, y]], not[subclass[u, x]],
          not[subclass[cart[x, y], union[S, inverse[S]]]], subclass[v, A[u]],
          subclass[A[u], v], u → intersection[x, image[S, set[w]]]]

Out[35]= or[not[member[v, y]], not[subclass[cart[x, y], union[S, inverse[S]]]],
           subclass[v, hull[x, w]], subclass[hull[x, w], v]] == True

In[36]:= (% /. {v → v_, w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

The set variables **v** and **w** are now removed:

```
In[37]:= Map[equal[0, composite[Id, complement[#]]] &, SubstTest[class, pair[w, v],
           or[not[member[v, x]], not[subclass[r, s]], subclass[v, hull[y, w]],
            subclass[hull[y, w], v]], {r → cart[y, x], s → union[S, inverse[S]]}]] // Reverse

Out[37]= or[not[subclass[cart[x, y], union[S, inverse[S]]]],
           subclass[cart[x, fix[HULL[y]]], union[S, inverse[S]]]] == True

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

A better result:

```
In[39]:= equiv[subclass[cart[x, fix[HULL[y]]], union[S, inverse[S]]],
          subclass[cart[x, y], union[S, inverse[S]]]]

Out[39]= True

In[40]:= subclass[cart[x_, fix[HULL[y_]]], union[S, inverse[S]]] :=
          subclass[cart[x, y], union[S, inverse[S]]]
```

Lemma.

```
In[41]:= Map[implies[subclass[cart[x, y], union[S, inverse[S]]], #] &,
          subclass[cart[y, x], union[S, inverse[S]]] // AssertTest]

Out[41]= or[not[subclass[cart[x, y], union[S, inverse[S]]]],
           subclass[cart[y, x], union[S, inverse[S]]]] == True

In[42]:= or[not[subclass[cart[x_, y_], union[S, inverse[S]]]],
           subclass[cart[y_, x_], union[S, inverse[S]]]] := True
```

Lemma.

```
In[43]:= SubstTest[implies, subclass[cart[u, v], union[S, inverse[S]]],
          subclass[cart[v, u], union[S, inverse[S]]], {u → fix[HULL[x]], v → y}]

Out[43]= or[not[subclass[cart[fix[HULL[x]], y], union[S, inverse[S]]]],
           subclass[cart[y, x], union[S, inverse[S]]]] == True

In[44]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[45]:= or[not[subclass[cart[x, y], union[S, inverse[S]]]],
           subclass[cart[fix[HULL[x]], y], union[S, inverse[S]]]] // AssertTest

Out[45]= or[not[subclass[cart[x, y], union[S, inverse[S]]]],
           subclass[cart[fix[HULL[x]], y], union[S, inverse[S]]]] == True

In[46]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem.

```
In[47]:= equiv[subclass[cart[fix[HULL[x]], y], union[S, inverse[S]]],
           subclass[cart[x, y], union[S, inverse[S]]]]

Out[47]= True

In[48]:= subclass[cart[fix[HULL[x_]], y_], union[S, inverse[S]]] :=
           subclass[cart[x, y], union[S, inverse[S]]]
```

Aclosure corollary

Lemma.

```
In[49]:= SubstTest[implies, subclass[cart[fix[HULL[x]], y], z],
           subclass[cart[Aclosure[x], y], z], z → union[S, inverse[S]]]

Out[49]= or[not[subclass[cart[x, y], union[S, inverse[S]]]],
            subclass[cart[Aclosure[x], y], union[S, inverse[S]]]] == True

In[50]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem.

```
In[51]:= equiv[subclass[cart[Aclosure[x], y], union[S, inverse[S]]],
           subclass[cart[x, y], union[S, inverse[S]]]]

Out[51]= True

In[52]:= subclass[cart[Aclosure[x_], y_], union[S, inverse[S]]] :=
           subclass[cart[x, y], union[S, inverse[S]]]
```

The companion result is now obtainable with **AssertTest**.

```
In[53]:= or[not[subclass[cart[x, y], union[S, inverse[S]]]],
            subclass[cart[x, Aclosure[y]], union[S, inverse[S]]]] // AssertTest

Out[53]= or[not[subclass[cart[x, y], union[S, inverse[S]]]],
            subclass[cart[x, Aclosure[y]], union[S, inverse[S]]]] == True

In[54]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed

In[55]:= equiv[subclass[cart[x, Aclosure[y]], union[S, inverse[S]]],
           subclass[cart[x, y], union[S, inverse[S]]]]

Out[55]= True

In[56]:= subclass[cart[x_, Aclosure[y_]], union[S, inverse[S]]] :=
           subclass[cart[x, y], union[S, inverse[S]]]
```

Variable-free corollary.

In[57]:=
```
image[inverse[ACLOSURE], cliques[union[S, inverse[S]]]] // Normality
```

Out[57]=
```
image[inverse[ACLOSURE], cliques[union[S, inverse[S]]]] == cliques[union[S, inverse[S]]]
```

In[58]:=
```
image[inverse[ACLOSURE], cliques[union[S, inverse[S]]]] :=
 cliques[union[S, inverse[S]]]
```