ADJECT[x]

Johan G. F. Belinfante
2004 March 23

```
In[1]:= << goedel55.23b ; << tools.m

        :Package Title: goedel55.23b           2004 March 23 at 5:25 p.m.

        It is now:  2004 Mar 24 at 10:20

        Loading Simplification Rules

        TOOLS.M                       Revised 2004 March 14

        weightlimit = 40
```

summary

This notebook is concerned with establishing necessary and sufficient conditions for a union of **ADJOIN** functions to be a partial order.

introduction

The function **ADJOIN[x]** is the set of all ordered pairs **pair[u, v]** such that **v = union[x, u]**. It is the function of a single variable obtained from the binary function **CUP** by fixing one of the two variables to be **x**. Since **CUP** is a symmetric function, it does not matter whether the variable that is fixed is the left variable or the right variable:

```
In[2]:= composite[CUP, LEFT[x]]

Out[2]= ADJOIN[x]

In[3]:= composite[CUP, RIGHT[x]]

Out[3]= ADJOIN[x]
```

If one replaces the single element **x** by a collection of elements, one obtains a relation that may be regarded as the "union" of a collection of **ADJOIN** functions:

```
In[4]:= class[pair[u, v], exists[w, and[member[w, x], equal[v, union[u, w]]]]]

Out[4]= composite[CUP, id[cart[V, x]], inverse[FIRST]]
```

For convenience, a temporary name will be given to such relations:

```
In[5]:= ADJECT[x_] := composite[CUP, id[cart[V, x]], inverse[FIRST]]
```

The following lemma is needed to establish the connection with **ADJOIN[x]**.

```
In[6]:= composite[inverse[RIGHT[x]], inverse[FIRST]] // DoubleInverse

Out[6]= composite[inverse[RIGHT[x]], inverse[FIRST]] == id[image[V, singleton[x]]]

In[7]:= composite[inverse[RIGHT[x_]], inverse[FIRST]] := id[image[V, singleton[x]]]
```

One has:

```
In[8]:= ADJECT[singleton[x]]

Out[8]= ADJOIN[x]
```

The relation **ADJECT[x]** is a "union" of **ADJOIN** functions in the following sense:

```
In[9]:= ADJECT[x] ==
         class[pair[u, v], exists[w, and[member[w, x], member[pair[u, v], ADJOIN[w]]]]]

Out[9]= True
```

ADJECT[FINITE]

In the notebook **TRV−K.NB** it was shown that the transitive closure of **union[Id, K]** is an example of such a relation, and this relation is a partial order:

```
In[10]:= ADJECT[FINITE]

Out[10]= union[Id, trv[K]]

In[11]:= PARTIALORDER[ADJECT[FINITE]]

Out[11]= True
```

Here **K** is the cover relation, consisting of all pairs **pair[u,v]** where **v** is obtained by adding a single element that does not belong to **u**.

```
In[12]:= class[pair[u, v], exists[w, and[equal[v, union[u, singleton[w]]], not[member[w, u]]]]]

Out[12]= K
```

If one leaves off the condition that the element **w** does not belong to **u**, one obtains:

```
In[13]:= class[pair[u, v], exists[w, equal[v, union[u, singleton[w]]]]]

Out[13]= union[K, id[complement[singleton[0]]]]
```

That is:

```
In[14]:= ADJECT[range[SINGLETON]]

Out[14]= union[K, id[complement[singleton[0]]]]
```

monotonicity

In this section it is shown that **ADJECT[x]** is a subclass of **ADJECT[y]** if and only if **x** is a subclass of **y**. The implication in one direction is obtained by examining the vertical section at the empty set:

```
In[15]:= SubstTest[implies, subclass[u, v], subclass[image[u, w], image[v, w]],
           {u -> composite[CUP, id[cart[V, x]], inverse[FIRST]],
          v -> composite[CUP, id[cart[V, y]], inverse[FIRST]], w -> singleton[0]}]

Out[15]= or[not[subclass[composite[CUP, id[cart[V, x]], inverse[FIRST]],
             composite[CUP, id[cart[V, y]], inverse[FIRST]]]], subclass[x, y]] == True

In[16]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

To obtain the inclusion in the opposite direction, one needs to know that **ADJECT[x]** can be written as the image of **x** under a suitable relation. One can discover this relation by using **abstract**, but the result at first seems complicated, until one rotates its inverse:

```
In[17]:= abstract[x, ADJECT[x]] // inverse // rotate

Out[17]= CUP
```

That is:

```
In[18]:= image[inverse[flip[rotate[CUP]]], x] == ADJECT[x]

Out[18]= True
```

With this knowledge, it is easy to derive the desired result as a special case of the monotonicity of **image**:

```
In[19]:= SubstTest[implies, subclass[x, y],
          subclass[image[w, x], image[w, y]], w -> inverse[flip[rotate[CUP]]]]

Out[19]= or[not[subclass[x, y]], subclass[composite[CUP, id[cart[V, x]], inverse[FIRST]],
            composite[CUP, id[cart[V, y]], inverse[FIRST]]]] == True

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

In[21]:= equiv[subclass[ADJECT[x], ADJECT[y]], subclass[x, y]]

Out[21]= True
```

Replacing **equiv** with **Equal**, one finds:

```
In[22]:= Equal[subclass[ADJECT[x], ADJECT[y]], subclass[x, y]]

Out[22]= subclass[composite[CUP, id[cart[V, x]], inverse[FIRST]],
           composite[CUP, id[cart[V, y]], inverse[FIRST]]] == subclass[x, y]
```

This justifies adding a new rewrite rule:

```
In[23]:= subclass[composite[CUP, id[cart[V, x_]], inverse[FIRST]],
           composite[CUP, id[cart[V, y_]], inverse[FIRST]]] := subclass[x, y]
```

Corollary.

```
In[24]:= equal[ADJECT[x], ADJECT[y]] // AssertTest

Out[24]= equal[composite[CUP, id[cart[V, x]], inverse[FIRST]],
           composite[CUP, id[cart[V, y]], inverse[FIRST]]] == equal[x, y]

In[25]:= equal[composite[CUP, id[cart[V, x_]], inverse[FIRST]],
           composite[CUP, id[cart[V, y_]], inverse[FIRST]]] := equal[x, y]
```

when is ADJECT[x] a partial order?

The theorem that **ADJECT[FINITE]** is a partial order raises the natural question, what are the conditions on **x** for **ADJECT[x]** to be a partial order? From the monotonicity property one finds:

```
In[26]:= SubstTest[subclass, composite[w, w], w, w -> ADJECT[x]] // Reverse

Out[26]= TRANSITIVE[composite[CUP, id[cart[V, x]], inverse[FIRST]]] ==
          subclass[image[CUP, cart[x, x]], x]
```

That is, **ADJECT[x]** is transitive if and only if **x** is closed under binary unions:

```
In[27]:= TRANSITIVE[composite[CUP, id[cart[V, x_]], inverse[FIRST]]] :=
          subclass[image[CUP, cart[x, x]], x]
```

The condition for **ADJECT[x]** to be reflexive is that **x** is either empty or holds the empty set as a member:

```
In[28]:= SubstTest[subclass, w, cart[fix[w], fix[w]], w -> ADJECT[x]] // Reverse

Out[28]= REFLEXIVE[composite[CUP, id[cart[V, x]], inverse[FIRST]]] ==
          or[equal[0, x], member[0, x]]

In[29]:= REFLEXIVE[composite[CUP, id[cart[V, x_]], inverse[FIRST]]] :=
          or[equal[0, x], member[0, x]]
```

The fact that **ADJECT[x]** is always antisymmetric follows from the following observation.

```
In[30]:= SubstTest[subclass, ADJECT[x], ADJECT[y], y -> V]

Out[30]= subclass[composite[CUP, id[cart[V, x]], inverse[FIRST]], S] == True

In[31]:= subclass[composite[CUP, id[cart[V, x_]], inverse[FIRST]], S] := True

In[32]:= Map[implies[#, subclass[x, Id]] &,
           SubstTest[subclass, x, intersection[u, v], {u -> S, v -> inverse[S]}] // Reverse]

Out[32]= or[not[subclass[x, S]], not[subclass[x, inverse[S]]], subclass[x, Id]] == True

In[33]:= (% /. x -> x_) /. Equal -> SetDelayed

In[34]:= SubstTest[implies, subclass[u, v],
           subclass[intersection[u, w], v], {u -> inverse[x], v -> inverse[S], w -> x}]

Out[34]= or[not[subclass[composite[Id, x], S]],
           subclass[intersection[x, inverse[x]], inverse[S]]] == True

In[35]:= (% /. x -> x_) /. Equal -> SetDelayed
```

```
In[36]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p1, p3], implies[p3, p4], not[implies[p1, p5]],
           {p1 -> subclass[x, S], p2 -> subclass[intersection[x, inverse[x]], S],
                 p3 -> subclass[composite[Id, x], S],
            p4 -> subclass[intersection[x, inverse[x]], inverse[S]],
            p5 -> subclass[intersection[x, inverse[x]], Id]}]]

Out[36]= or[not[subclass[x, S]], subclass[intersection[x, inverse[x]], Id]] == True

In[37]:= or[not[subclass[x_, S]], subclass[intersection[x_, inverse[x_]], Id]] := True

In[38]:= SubstTest[implies, subclass[w, S],
          subclass[intersection[w, inverse[w]], Id], w -> ADJECT[x]]

Out[38]= subclass[intersection[composite[CUP, id[cart[V, x]], inverse[FIRST]],
           composite[FIRST, id[cart[V, x]], inverse[CUP]]], Id] == True

In[39]:= (% /. x -> x_) /. Equal -> SetDelayed

In[40]:= SubstTest[and, REFLEXIVE[w], ANTISYMMETRIC[w], TRANSITIVE[w], w -> ADJECT[x]] // Reverse

Out[40]= PARTIALORDER[composite[CUP, id[cart[V, x]], inverse[FIRST]]] ==
          or[and[member[0, x], subclass[image[CUP, cart[x, x]], x]], equal[0, x]]
```

It follows that **ADJECT[x]** is a partial order if either **x** is the empty set, or else **x** holds the empty set and is closed under binary unions:

```
PARTIALORDER[composite[CUP, id[cart[V, x]], inverse[FIRST]]] ==
 or[and[member[0, x], subclass[image[CUP, cart[x, x]], x]], equal[0, x]]

In[41]:= PARTIALORDER[composite[CUP, id[cart[V, x_]], inverse[FIRST]]] :=
          or[and[member[0, x], subclass[image[CUP, cart[x, x]], x]], equal[0, x]]
```

when is ADJECT[x] a function?

This section contains a derivation of the unsurprising fact that **ADJECT[x]** is a function if and only if **x** is either empty or a singleton. To do this, one needs some facts about functions. The first step is the following lemma about APPLY that holds whether or not **z** is a function:

```
In[42]:= SubstTest[implies, and[equal[u, v], member[v, w]], member[u, w],
          {u -> y, v -> singleton[APPLY[z, x]], w -> range[SINGLETON]}]

Out[42]= or[member[y, range[SINGLETON]],
           not[equal[y, singleton[APPLY[z, x]]]], not[member[x, domain[z]]]] == True

In[43]:= or[member[y_, range[SINGLETON]],
           not[equal[y_, singleton[APPLY[z_, x_]]]], not[member[x_, domain[z_]]]] := True
```

This lemma is used to show the obvious fact that vertical sections of functions at points of the domain are singletons:

```
In[44]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[and[p1, p2], p4], implies[and[p3, p4], p5],
           not[implies[and[p1, p2], p5]], {p1 -> FUNCTION[x], p2 -> member[y, domain[x]],
            p3 -> equal[image[x, singleton[y]], singleton[APPLY[x, y]]],
          p4 -> member[APPLY[x, y], V],
             p5 -> member[image[x, singleton[y]], range[SINGLETON]]}]]

Out[44]= or[member[image[x, singleton[y]], range[SINGLETON]],
           not[FUNCTION[x]], not[member[y, domain[x]]]] == True
```

```
In[45]:= or[member[image[x_, singleton[y_]], range[SINGLETON]],
            not[FUNCTION[x_]], not[member[y_, domain[x_]]]] := True
```

Applying this to the case of the vertical section of **ADJECT[x]** at the empty set yields:

```
In[46]:= SubstTest[implies, and[member[w, domain[z]], FUNCTION[z]],
           member[image[z, singleton[w]], range[SINGLETON]],
              {z -> ADJECT[x], w -> 0}]

Out[46]= or[equal[0, x], member[x, range[SINGLETON]],
            not[FUNCTION[composite[CUP, id[cart[V, x]], inverse[FIRST]]]]] == True

In[47]:= (% /. x -> x_) /. Equal -> SetDelayed
```

The converse statement requires showing that **ADJECT[x]** is a function when **x** is the empty set or a singleton. The case of the empty set is done this way:

```
In[48]:= SubstTest[implies, and[FUNCTION[u], equal[u, v]], FUNCTION[v], {u -> 0, v -> ADJECT[x]}]

Out[48]= or[FUNCTION[composite[CUP, id[cart[V, x]], inverse[FIRST]]], not[equal[0, x]]] == True

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

Lemma.

```
In[50]:= SubstTest[equal, ADJECT[y], ADJECT[z], z -> singleton[x]]

Out[50]= equal[ADJOIN[x], composite[CUP, id[cart[V, y]], inverse[FIRST]]] ==
           equal[y, singleton[x]]

In[51]:= equal[ADJOIN[x_], composite[CUP, id[cart[V, y_]], inverse[FIRST]]] :=
           equal[y, singleton[x]]
```

Lemma.

```
In[52]:= SubstTest[implies, and[FUNCTION[u], equal[u, v]],
           FUNCTION[v], {u -> ADJOIN[y], v -> ADJECT[x]}]

Out[52]= or[FUNCTION[composite[CUP, id[cart[V, x]], inverse[FIRST]]],
            not[equal[x, singleton[y]]]] == True

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

The remaining problem is to eliminate the variable **y** using **class**. To prevent the **FUNCTION** hypothesis from being rewritten by the action of **class**, one needs the following normalization rule.

```
In[54]:= SubstTest[subclass, composite[y, inverse[y]], Id, y -> ADJECT[x]]

Out[54]= subclass[composite[CUP, id[cart[V, x]],
             inverse[FIRST], FIRST, id[cart[V, x]], inverse[CUP]], Id] ==
           FUNCTION[composite[CUP, id[cart[V, x]], inverse[FIRST]]]
```

This normalization rule is added as a temporary rewrite rule.

```
In[55]:= subclass[composite[CUP, id[cart[V, x_]],
             inverse[FIRST], FIRST, id[cart[V, x_]], inverse[CUP]], Id] :=
           FUNCTION[composite[CUP, id[cart[V, x]], inverse[FIRST]]]
```

The elimination of the variable **y** is done as follows:

```
In[56]:= Map[equal[V, #] &, SubstTest[class, y,
           or[FUNCTION[z], not[equal[x, singleton[y]]]], z -> ADJECT[x]]] // Reverse

Out[56]= or[FUNCTION[composite[CUP, id[cart[V, x]], inverse[FIRST]]],
           not[member[x, range[SINGLETON]]]] == True

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

All that remains is to use double negation to combine the three clauses into a logical equivalence assertion:

```
In[58]:= equiv[FUNCTION[composite[CUP, id[cart[V, x]], inverse[FIRST]]],
           or[equal[0, x], member[x, range[SINGLETON]]]] // not // not

Out[58]= True
```

This justifies the following rewrite rule:

```
In[59]:= FUNCTION[composite[CUP, id[cart[V, x_]], inverse[FIRST]]] :=
          or[equal[0, x], member[x, range[SINGLETON]]]
```