compactness is not changed by adding or removing 0

Johan G. F. Belinfante
2004 October 10

```
In[1]:= SetDirectory["i:"]; << goedel62.09a; << tools.m

        :Package Title: goedel62.09a          2004 October 9 at 11:25 a.m.

        It is now:  2004 Oct 10 at 15:51

        Loading Simplification Rules

        TOOLS.M                      Revised 2004 September 25

        weightlimit = 40
```

---

summary

Adding or removing the empty set from a collection of sets does not affect its compactness.

---

cutting out {0}

In this section it is shown that if **x** is compact, then removing the empty set does not affect compactness. To start, we need the fact that the new collection is coarser than the original one:

```
In[2]:= Map[equal[0, #] &,
         dif[IMAGE[id[complement[singleton[0]]]], inverse[COARSER]] // VSNormality]

Out[2]= subclass[IMAGE[id[complement[singleton[0]]]], inverse[COARSER]] == True

In[3]:= subclass[IMAGE[id[complement[singleton[0]]]], inverse[COARSER]] := True
```

The compactness result is first formulated in variable-free form:

```
In[4]:= SubstTest[implies, subclass[u, v], subclass[image[u, w], image[v, w]],
         {u → IMAGE[id[complement[singleton[0]]]], v → inverse[COARSER], w → COMPACT}]

Out[4]= subclass[image[IMAGE[id[complement[singleton[0]]]], COMPACT], COMPACT] == True

In[5]:= % /. Equal → SetDelayed
```

The interpretation of the result is easier to comprehend when a variable is introduced.

```
In[6]:= SubstTest[implies,
         and[member[x, y], subclass[y, z]], member[x, z], {y -> COMPACT,
          z -> image[inverse[IMAGE[id[complement[singleton[0]]]]], COMPACT]}]

Out[6]= or[member[intersection[x, complement[singleton[0]]], COMPACT],
          not[member[x, COMPACT]]] == True

In[7]:= (% /. x → x_) /. Equal → SetDelayed
```

These temporary results will be improved upon shortly.

---

adding in {0}

The adjunction result is similar:

```
In[8]:= SubstTest[implies, subclass[u, v],
         subclass[image[inverse[u], w], image[inverse[v], w]],
         {u → ADJOIN[singleton[0]], v → COARSER, w → COMPACT}]

Out[8]= subclass[image[inverse[ADJOIN[singleton[0]]], COMPACT], COMPACT] == True

In[9]:= % /. Equal → SetDelayed
```

Introducing a variable helps clarify this result.

```
In[10]:= SubstTest[implies, and[member[x, y], subclass[y, z]], member[x, z],
          {y -> image[inverse[ADJOIN[singleton[0]]], COMPACT], z -> COMPACT}]

Out[10]= or[member[x, COMPACT], not[member[union[x, singleton[0]], COMPACT]]] == True

In[11]:= (% /. x → x_) /. Equal → SetDelayed
```

This result is similar to the one derived in the preceding section in that a compact collection is being trimmed down to produce another compact collection. In the next section the converse statement will be derived.

an inverse implication

The idea: Suppose **x** is compact, and one wants to show that **union[x,singleton[0]]** is also compact. Let **y** be any collection that is coarser than **union[x,singleton[0]]**. What one needs to show is that this collection is coarser than a finite one. There are two cases to be considered: either the empty set **0** belongs to **y** or not. If not, then **y** is coarser than **x**, and singce the latter is compact, **y** must in this case be coarser than a finite set. If **0** does belong to **y**, then **dif[y, singleton[0]]** is coarser than **x**, and so **dif[y,singleton[0]]** is coarser than a finite set. But then so is **y**. To carry out this argument one needs this class:

```
In[12]:= class[y, member[pair[dif[y, singleton[0]], x], z]] /. z → COARSER

Out[12]= image[inverse[IMAGE[id[complement[singleton[0]]]]],
          image[inverse[COARSER], singleton[x]]]
```

The folowing takes 37 seconds with the **simplify** and **cond** flags turned on (the default choice). Turning off the **simplify** flag reduces this to 2 seconds.

```
In[13]:= simplify = False;

In[14]:= Map[equal[0, #] &,
          dif[image[inverse[COARSER], singleton[union[x, singleton[0]]]],
            union[image[inverse[COARSER], singleton[x]],
             image[inverse[IMAGE[id[complement[singleton[0]]]]],
              image[inverse[COARSER], singleton[x]]]]] // Renormality]

Out[14]= subclass[image[inverse[COARSER], singleton[union[x, singleton[0]]]],
           union[image[inverse[COARSER], singleton[x]],
            image[inverse[IMAGE[id[complement[singleton[0]]]]],
             image[inverse[COARSER], singleton[x]]]]] == True

In[15]:= (% /. x → x_) /. Equal → SetDelayed
```

Technical lemma.

```
In[16]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
          {u -> image[inverse[COARSER], singleton[union[x, singleton[0]]]],
           v -> union[image[inverse[COARSER], singleton[x]],
             image[inverse[IMAGE[id[complement[singleton[0]]]]],
              image[inverse[COARSER], singleton[x]]]], w → image[COARSER, FINITE]}]

Out[16]= or[member[union[x, singleton[0]], COMPACT], not[member[x, COMPACT]],
           not[subclass[image[inverse[IMAGE[id[complement[singleton[0]]]]],
              image[inverse[COARSER], singleton[x]]], image[COARSER, FINITE]]]] == True

In[17]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[18]:= Map[implies[member[x, COMPACT], #] &,
          SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
           {u → image[inverse[COARSER], singleton[x]], v → image[COARSER, FINITE],
            w -> inverse[IMAGE[id[complement[singleton[0]]]]]}]]

Out[18]= or[not[member[x, COMPACT]],
           subclass[intersection[image[inverse[COARSER], singleton[x]],
             P[complement[singleton[0]]]], image[COARSER, FINITE]]] == True

In[19]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[20]:= SubstTest[subclass, inverse[x], inverse[y],
          {x → IMAGE[id[complement[singleton[0]]]], y → inverse[COARSER]}]

Out[20]= subclass[inverse[IMAGE[id[complement[singleton[0]]]]], COARSER] == True

In[21]:= subclass[inverse[IMAGE[id[complement[singleton[0]]]]], COARSER] := True

In[22]:= SubstTest[implies, subclass[u, v], subclass[image[u, w], image[v, w]],
          {u -> inverse[IMAGE[id[complement[singleton[0]]]]],
           v -> COARSER, w → image[COARSER, x]}]

Out[22]= subclass[image[inverse[IMAGE[id[complement[singleton[0]]]]],
            image[COARSER, x]], image[COARSER, x]] == True

In[23]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[24]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u -> image[inverse[IMAGE[id[complement[singleton[0]]]]],
              image[inverse[COARSER], singleton[x]]],
            v -> image[inverse[IMAGE[id[complement[singleton[0]]]]],
              image[COARSER, FINITE]],
            w -> image[COARSER, FINITE]}] /. w ->
           inverse[IMAGE[id[complement[singleton[0]]]]]

Out[24]= or[not[subclass[intersection[image[inverse[COARSER], singleton[x]],
              P[complement[singleton[0]]]], image[COARSER, FINITE]]],
           subclass[image[inverse[IMAGE[id[complement[singleton[0]]]]],
             image[inverse[COARSER], singleton[x]]], image[COARSER, FINITE]]] == True

In[25]:= (% /. x → x_) /. Equal → SetDelayed
```

The main theorem:

```
In[26]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p2, p3], implies[and[p1, p3], p4], not[implies[p1, p4]],
           {p1 → member[x, COMPACT],
            p2 -> subclass[intersection[image[inverse[COARSER], singleton[x]],
              P[complement[singleton[0]]]], image[COARSER, FINITE]],
            p3 -> subclass[image[inverse[IMAGE[id[complement[singleton[0]]]]],
              image[inverse[COARSER], singleton[x]]], image[COARSER, FINITE]],
            p4 -> member[union[x, singleton[0]], COMPACT]}]]

Out[26]= or[member[union[x, singleton[0]], COMPACT], not[member[x, COMPACT]]] == True

In[27]:= (% /. x → x_) /. Equal → SetDelayed
```

Combining this result with that obtained in the preceding section yields a logical equivalence:

```
In[28]:= equiv[member[union[x, singleton[0]], COMPACT], member[x, COMPACT]]

Out[28]= True

In[29]:= member[union[x_, singleton[0]], COMPACT] := member[x, COMPACT]
```

The following corollary is obtained for the case of removing the empty set.

```
In[30]:= SubstTest[member, union[y, singleton[0]],
           COMPACT, y → dif[x, singleton[0]]] // Reverse

Out[30]= member[intersection[x, complement[singleton[0]]], COMPACT] ==
          member[x, COMPACT]

In[31]:= member[intersection[x_, complement[singleton[0]]], COMPACT] :=
          member[x, COMPACT]
```

variable-free equations

```
In[32]:= image[inverse[ADJOIN[singleton[0]]], COMPACT] // Normality
Out[32]= image[inverse[ADJOIN[singleton[0]]], COMPACT] == COMPACT

In[33]:= image[inverse[ADJOIN[singleton[0]]], COMPACT] := COMPACT

In[34]:= image[inverse[IMAGE[id[complement[singleton[0]]]]], COMPACT] // Normality
Out[34]= image[inverse[IMAGE[id[complement[singleton[0]]]]], COMPACT] == COMPACT

In[35]:= image[inverse[IMAGE[id[complement[singleton[0]]]]], COMPACT] := COMPACT

In[36]:= ImageComp[ADJOIN[singleton[0]],
           inverse[ADJOIN[singleton[0]]], COMPACT] // Reverse
Out[36]= image[ADJOIN[singleton[0]], COMPACT] ==
          intersection[COMPACT, complement[P[complement[singleton[0]]]]]

In[37]:= image[ADJOIN[singleton[0]], COMPACT] :=
          intersection[COMPACT, complement[P[complement[singleton[0]]]]]

In[38]:= ImageComp[IMAGE[id[complement[singleton[0]]]],
           inverse[IMAGE[id[complement[singleton[0]]]]], COMPACT] // Reverse
Out[38]= image[IMAGE[id[complement[singleton[0]]]], COMPACT] ==
          intersection[COMPACT, P[complement[singleton[0]]]]

In[39]:= image[IMAGE[id[complement[singleton[0]]]], COMPACT] :=
          intersection[COMPACT, P[complement[singleton[0]]]]
```