T2 separation condition

Johan G. F. Belinfante
2003 December 30

```
In[1]:= << goedel52.t40; << tools.m

        :Package Title: goedel52.t40      2003 December 30 at 12:10 noon

        It is now:  2004 Jan 30 at 8:40

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 November 15

        weightlimit = 40
```

summary

To facilitate reasoning about **T2**, the membership rule for **T2** has been wrapped with **class**. The rewrite rules for this class are re−examined in this notebook. As an application, the cofinite topology on the natural numbers is used to show that **T1** does not imply **T2**.

membership clauses for the T2 separation condition

The results in this section can be derived more quickly if the **simplify** flag is turned off.

```
In[2]:= simplify = False;
```

It is convenient to have available clauses characterizing membership in the class **T2**.

```
In[3]:= Map[implies[member[x, T2], #] &, member[x, T2] // AssertTest] // MapNotNot // Reverse

Out[3]= or[not[member[x, T2]], subclass[composite[id[U[x]], Di, id[U[x]]],
          composite[inverse[E], id[x], DISJOINT, id[x], E]]] == True

In[4]:= or[not[member[x_, T2]], subclass[composite[id[U[x_]], Di, id[U[x_]]],
          composite[inverse[E], id[x_], DISJOINT, id[x_], E]]] := True

In[5]:= Map[implies[and[member[x, y], #], member[x, T2]] &,
          member[x, T2] // AssertTest] // Reverse

Out[5]= or[member[x, T2], not[member[x, y]], not[subclass[composite[id[U[x]], Di, id[U[x]]],
           composite[inverse[E], id[x], DISJOINT, id[x], E]]]] == True

In[6]:= or[member[x_, T2], not[member[x_, y_]],
          not[subclass[composite[id[U[x_]], Di, id[U[x_]]],
            composite[inverse[E], id[x_], DISJOINT, id[x_], E]]]] := True
```

lemma

The following lemma will be used to derive a Uclosure property of the class **T2** that is analogous to one derived for the class **T1**.

```
In[7]:= composite[DISJOINT, id[Uclosure[x]], E] // RelnNormality

Out[7]= composite[DISJOINT, id[Uclosure[x]], E] == composite[DISJOINT, id[x], E]

In[8]:= composite[DISJOINT, id[Uclosure[x_]], E] := composite[DISJOINT, id[x], E]
```

The inverse rule is also needed:

```
In[9]:= composite[inverse[E], id[Uclosure[x]], DISJOINT] // DoubleInverse

Out[9]= composite[inverse[E], id[Uclosure[x]], DISJOINT] ==
         composite[inverse[E], id[x], DISJOINT]

In[10]:= composite[inverse[E], id[Uclosure[x_]], DISJOINT] :=
          composite[inverse[E], id[x], DISJOINT]
```

Although it was not needed, the following corollary helps explain why the class **T1** satisfies a similar property.

```
In[11]:= Assoc[inverse[E], DISJOINT, composite[id[Uclosure[x]], E]] // Reverse

Out[11]= composite[complement[inverse[E]], id[Uclosure[x]], E] ==
          composite[complement[inverse[E]], id[x], E]

In[12]:= composite[complement[inverse[E]], id[Uclosure[x_]], E] :=
          composite[complement[inverse[E]], id[x], E]

In[13]:= composite[inverse[E], id[Uclosure[x]], complement[E]] // DoubleInverse

Out[13]= composite[inverse[E], id[Uclosure[x]], complement[E]] ==
          composite[inverse[E], id[x], complement[E]]

In[14]:= composite[inverse[E], id[Uclosure[x_]], complement[E]] :=
          composite[inverse[E], id[x], complement[E]]
```

Uclosure theorem

The Uclosure theorem for **T2** resembles that for **T1**. It is most easily derived in variable−free form:

```
In[15]:= image[inverse[UCLOSURE], T2] // Normality

Out[15]= image[inverse[UCLOSURE], T2] == T2

In[16]:= image[inverse[UCLOSURE], T2] := T2
```

Corollary:

```
In[17]:= ImageComp[UCLOSURE, inverse[UCLOSURE], T2] // Reverse

Out[17]= image[UCLOSURE, T2] == intersection[T2, fix[UCLOSURE]]
```

```
In[18]:= image[UCLOSURE, T2] := intersection[T2, fix[UCLOSURE]]
```

A version with a variable can be obtained from the variable−free version:

```
In[19]:= SubstTest[member, x, image[inverse[UCLOSURE], y], y -> T2] // Reverse

Out[19]= member[Uclosure[x], T2] == member[x, T2]

In[20]:= member[Uclosure[x_], T2] := member[x, T2]
```

examples

```
In[21]:= member[0, T2] // AssertTest

Out[21]= member[0, T2] == True

In[22]:= member[0, T2] := True
```

Corollary:

```
In[23]:= equal[image[S, T2], V]

Out[23]= True

In[24]:= image[S, T2] := V
```

The following rules are about the singleton case.

```
In[25]:= member[singleton[x], T2] // AssertTest

Out[25]= member[singleton[x], T2] ==
          or[equal[0, x], member[x, range[SINGLETON]], not[member[x, V]]]

In[26]:= member[singleton[x_], T2] :=
          or[equal[0, x], member[x, range[SINGLETON]], not[member[x, V]]]

In[27]:= image[inverse[SINGLETON], T2] // Normality

Out[27]= image[inverse[SINGLETON], T2] == union[range[SINGLETON], singleton[0]]

In[28]:= image[inverse[SINGLETON], T2] := union[range[SINGLETON], singleton[0]]
```

The trivial topology is **pairset[0, x]**. The following rule says that the trivial topolgy is **T2** only when the space does not have two distinct elements.

```
In[29]:= member[pairset[0, x], T2] // AssertTest

Out[29]= member[pairset[0, x], T2] ==
          or[equal[0, x], member[x, range[SINGLETON]], not[member[x, V]]]

In[30]:= member[pairset[0, x_], T2] :=
          or[equal[0, x], member[x, range[SINGLETON]], not[member[x, V]]]
```

the discrete topology is T2

The discrete topology always satisfies **T2**.

```
In[31]:= member[P[x], T2] // AssertTest
Out[31]= member[P[x], T2] == member[x, V]

In[32]:= member[P[x_], T2] := member[x, V]
```

The following is a special case.

```
In[33]:= member[succ[singleton[0]], T2] // AssertTest
Out[33]= member[succ[singleton[0]], T2] == True

In[34]:= member[succ[singleton[0]], T2] := True
```

The following formula is a restatement of the fact that any set can be given a T2 topology (namely the discrete topology).

```
In[35]:= SubstTest[implies, subclass[u, v],
           subclass[image[BIGCUP, u], image[BIGCUP, v]], {u -> range[POWER], v -> T2}]
Out[35]= subclass[V, image[BIGCUP, T2]] == True

In[36]:= % /. Equal -> SetDelayed

In[37]:= equal[V, image[BIGCUP, T2]] // AssertTest
Out[37]= equal[V, image[BIGCUP, T2]] == True

In[38]:= image[BIGCUP, T2] := V
```

adjoining 0

```
In[39]:= image[inverse[ADJOIN[singleton[0]]], T2] // Normality
Out[39]= image[inverse[ADJOIN[singleton[0]]], T2] == T2

In[40]:= image[inverse[ADJOIN[singleton[0]]], T2] := T2

In[41]:= SubstTest[member, x, image[inverse[ADJOIN[singleton[0]]], y], y -> T2] // Reverse
Out[41]= member[union[x, singleton[0]], T2] == member[x, T2]

In[42]:= member[union[x_, singleton[0]], T2] := member[x, T2]
```

some counterexamples

```
In[43]:= Map[not, SubstTest[implies, and[member[x, y], subclass[y, z]],
           member[x, z], {x -> 0, y -> T2, z -> fix[UCLOSURE]}]]

Out[43]= subclass[T2, fix[UCLOSURE]] == False

In[44]:= subclass[T2, fix[UCLOSURE]] := False

In[45]:= member[pairset[singleton[x], singleton[0]], T2] // AssertTest

Out[45]= member[pairset[singleton[0], singleton[x]], T2] == True

In[46]:= member[pairset[singleton[0], singleton[x_]], T2] := True

In[47]:= Map[not, SubstTest[implies, and[member[t, u], subclass[u, v]], member[t, v],
             {t -> pairset[singleton[0], singleton[x]], u -> T2, v -> binclosed[CAP]}]] /.
           x -> singleton[0]

Out[47]= subclass[T2, binclosed[CAP]] == False

In[48]:= subclass[T2, binclosed[CAP]] := False
```

The existence of non−T2 topologies with more than one element can be established as follows:

```
In[49]:= member[succ[succ[singleton[0]]], T2] // AssertTest

Out[49]= member[succ[succ[singleton[0]]], T2] == False

In[50]:= member[succ[succ[singleton[0]]], T2] := False

In[51]:= Map[not, SubstTest[implies, and[member[t, u], subclass[u, v]], member[t, v],
             {t -> succ[succ[singleton[0]]], u -> TOPS, v -> T2}]]

Out[51]= subclass[TOPS, T2] == False

In[52]:= subclass[TOPS, T2] := False
```

COARSER formula

In this section a rule is derived which asserts that any refinement of a **T2** collection also belongs to **T2**. An equational substitution lemma is needed.

```
In[53]:= SubstTest[implies, equal[u, v], equal[image[u, w], image[v, w]],
           {u -> id[cart[x, x]], v -> id[cart[y, y]]}]

Out[53]= or[equal[composite[id[x], w, id[x]], composite[id[y], w, id[y]]], not[equal[x, y]]] ==
           True

In[54]:= (% /. {w -> w_, x -> x_, y -> y_}) /. Equal -> SetDelayed
```

Composites are images of cross products and images are monotone.

```
In[55]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
           {u -> composite[id[x], DISJOINT, id[x]],
          v -> composite[id[y], DISJOINT, id[y]], w -> cross[inverse[E], inverse[E]]}]

Out[55]= or[not[subclass[intersection[x, image[DISJOINT, x]], y]],
           subclass[composite[inverse[E], id[x], DISJOINT, id[x], E],
            composite[inverse[E], id[y], DISJOINT, id[y], E]]] == True

In[56]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

Several more applications of equality substitution are needed.

```
In[57]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
           implies[p3, p4], implies[and[p2, p5], p6], implies[and[p4, p6], p7],
             not[implies[and[p1, p5], p7]], {p1 -> member[pair[x, y], COARSER],
            p2 -> equal[composite[id[U[x]], Di, id[U[x]]], composite[id[U[y]], Di, id[U[y]]]],
                 p3 -> subclass[composite[id[x], DISJOINT, id[x]],
               composite[id[y], DISJOINT, id[y]]],
         p4 -> subclass[composite[inverse[E], id[x], DISJOINT, id[x], E],
               composite[inverse[E], id[y], DISJOINT, id[y], E]],
                  p5 -> subclass[composite[id[U[x]], Di, id[U[x]]],
               composite[inverse[E], id[x], DISJOINT, id[x], E]],
                        p6 -> subclass[composite[id[U[y]], Di, id[U[y]]],
               composite[inverse[E], id[x], DISJOINT, id[x], E]],
                  p7 -> subclass[composite[id[U[y]], Di, id[U[y]]],
               composite[inverse[E], id[y], DISJOINT, id[y], E]}]]

Out[57]= or[not[equal[U[x], U[y]]], not[member[y, V]],
           not[subclass[x, y]], not[subclass[composite[id[U[x]], Di, id[U[x]]],
             composite[inverse[E], id[x], DISJOINT, id[x], E]]],
           subclass[composite[id[U[y]], Di, id[U[y]]],
            composite[inverse[E], id[y], DISJOINT, id[y], E]]] == True

In[58]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

It follows that if **x** belongs to **T2**, and if **x** is coarser than **y**, then **y** belongs to **T2**.

```
In[59]:= Map[not, SubstTest[and, implies[q1, p5],
           implies[p1, q2], implies[and[q2, p7], q3], implies[and[p1, p5], p7],
             not[implies[and[p1, q1], q3]],
           {q1 -> member[x, T2], q2 -> member[y, V], q3 -> member[y, T2],
                  p1 -> member[pair[x, y], COARSER],
                 p5 -> subclass[composite[id[U[x]], Di, id[U[x]]],
               composite[inverse[E], id[x], DISJOINT, id[x], E]],
                  p7 -> subclass[composite[id[U[y]], Di, id[U[y]]],
               composite[inverse[E], id[y], DISJOINT, id[y], E]]}]]

Out[59]= or[member[y, T2], not[equal[U[x], U[y]]],
           not[member[x, T2]], not[member[y, V]], not[subclass[x, y]]] == True

In[60]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

The variables can be removed.

```
In[61]:= Map[equal[0, composite[Id, complement[#]]] &, SubstTest[class, pair[x, y], implies[
           and[member[pair[x, y], z], member[x, T2]], member[y, T2]], z -> COARSER]] // Reverse

Out[61]= subclass[image[COARSER, T2], T2] == True

In[62]:= % /. Equal -> SetDelayed
```

Since the opposite inclusion also holds, an equation is obtained.

```
In[63]:= SubstTest[and, subclass[u, v], subclass[v, u], {u -> T2, v -> image[COARSER, T2]}]

Out[63]= True == equal[T2, image[COARSER, T2]]

In[64]:= image[COARSER, T2] := T2
```

strategy for the proof that T1 does not imply T2

The set **image[RC[omega],FINITE]** of cofinite subsets of **omega** is a basis for a topology on **omega**. The cofinite topology itself is obtained by adding the empty set to this collection. It will be shown below that the collection of cofinite subsets of **omega** belongs to **T1**, but does not belong to **T2**. The proof that this collection belongs to **T1** will follow from a characterization of membership in the class **T1** that follows from a corresponding characterization for the class **PointClosed**.

characterization of T1 in terms of point−closedness

A collection **t** belongs to **T1** if and only if **Uclosure[t]** belongs to **PointClosed**. Consequently, one can obtain a characterization of membership in **T1** from the corresponding conditions for **PointClosed**.

```
In[65]:= SubstTest[implies, member[t, PointClosed],
          subclass[image[SINGLETON, U[t]], image[RC[U[t]], t]], t -> Uclosure[x]]

Out[65]= or[not[member[x, T1]],
           subclass[image[SINGLETON, U[x]], image[RC[U[x]], Uclosure[x]]]] == True

In[66]:= or[not[member[x_, T1]],
           subclass[image[SINGLETON, U[x_]], image[RC[U[x_]], Uclosure[x_]]]] := True

In[67]:= Map[implies[member[x, y], #] &, SubstTest[implies,
           and[member[t, V], subclass[image[SINGLETON, U[t]], image[RC[U[t]], t]]],
           member[t, PointClosed], t -> Uclosure[x]]]

Out[67]= or[member[x, T1], not[member[x, y]],
           not[subclass[image[SINGLETON, U[x]], image[RC[U[x]], Uclosure[x]]]]] == True

In[68]:= or[member[x_, T1], not[member[x_, y_]],
           not[subclass[image[SINGLETON, U[x_]], image[RC[U[x_]], Uclosure[x_]]]]] := True
```

the set of cofinite subsets of omega belongs to T1

```
In[69]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
             {u -> image[SINGLETON, x], v -> range[SINGLETON], w -> FINITE}]

Out[69]= subclass[image[SINGLETON, x], FINITE] == True

In[70]:= subclass[image[SINGLETON, x_], FINITE] := True

In[71]:= SubstTest[subclass, u, intersection[v, w],
          {u -> image[SINGLETON, omega], v -> FINITE, w -> P[omega]}]

Out[71]= subclass[image[SINGLETON, omega], image[inverse[S], omega]] == True
```

```
In[72]:= subclass[image[SINGLETON, omega], image[inverse[S], omega]] := True

In[73]:= SubstTest[implies,
          and[member[x, V], subclass[image[SINGLETON, U[x]], image[RC[U[x]], Uclosure[x]]] ],
          member[x, T1], x -> image[RC[omega], FINITE]]

Out[73]= member[image[RC[omega], FINITE], T1] == True

In[74]:= member[image[RC[omega], FINITE], T1] := True
```

the set of cofinite subsets of omega does not belong to T2

Lemma.

```
In[75]:= Assoc[IMAGE[id[x]], id[P[x]], id[image[RC[x], y]]]

Out[75]= composite[IMAGE[id[x]], id[image[RC[x], y]]] == id[image[RC[x], y]]

In[76]:= composite[IMAGE[id[x_]], id[image[RC[x_], y_]]] := id[image[RC[x], y]]
```

The idea in the following derivation is to make use of the fact that the union of two sets is finite if and only if the individual sets are finite. This is first translated into a result about cofinite sets using complementation, and then the variables are eliminated. All this can be done simultaneously.

```
In[77]:= Map[
          class[pair[x, y], and[member[x, P[omega]], member[y, P[omega]], not[#]]] &, Map[assert,
           SubstTest[implies, and[equal[u, union[s, t]], member[union[s, t], w]], member[u, w],
          {u -> omega, w -> FINITE, s -> dif[omega, x], t -> dif[omega, y]}]]]

Out[77]= composite[id[image[RC[omega], FINITE]], DISJOINT, id[image[RC[omega], FINITE]]] == 0

In[78]:= composite[id[image[RC[omega], FINITE]], DISJOINT, id[image[RC[omega], FINITE]]] := 0
```

Lemma.

```
In[79]:= member[omega, range[SINGLETON]] // AssertTest

Out[79]= member[omega, range[SINGLETON]] == False

In[80]:= member[omega, range[SINGLETON]] := False

In[81]:= subclass[composite[id[omega], Di, id[omega]], 0] // AssertTest

Out[81]= subclass[composite[id[omega], Di, id[omega]], 0] == False

In[82]:= subclass[composite[id[omega], Di, id[omega]], 0] := False

In[83]:= Map[not, SubstTest[implies, member[t, T2], subclass[composite[id[U[t]], Di, id[U[t]]],
            composite[inverse[E], id[t], DISJOINT, id[t], E]],
             t -> image[RC[omega], FINITE]]]

Out[83]= member[image[RC[omega], FINITE], T2] == False

In[84]:= member[image[RC[omega], FINITE], T2] := False
```

T1 does not imply T2

Since the collection of cofinite subsets of omega belongs to **T1** but not **T2**, it follows that **T2** is not contained in **T1**.

```
In[85]:= Map[not, SubstTest[implies, and[member[x, y], subclass[y, z]], member[x, z],
             {x -> image[RC[omega], FINITE], y -> T1, z -> T2}]]

Out[85]= subclass[T1, T2] == False

In[86]:= subclass[T1, T2] := False
```

The set **image[RC[omega], FINITE]** is not itself a topology, but it is the basis for a topology. One goes from the basis to the topology by applyiing **Uclosure**. In the present case, this amounts to just adding the empty set to the basis to get the topology.

```
In[87]:= Map[not, SubstTest[implies, and[member[x, y], subclass[y, z]], member[x, z],
             {x -> Uclosure[image[RC[omega], FINITE]], y -> intersection[T1, TOPS], z -> T2}]]

Out[87]= subclass[intersection[T1, TOPS], T2] == False

In[88]:= subclass[intersection[T1, TOPS], T2] := False
```

appendix: some more Uclosure results

The following further formulas were not needed in this notebook, but are included because their derivation is analogous to ones that were used.

```
In[89]:= Assoc[E, complement[inverse[E]], composite[id[Uclosure[x]], E]] // Reverse

Out[89]= composite[complement[inverse[S]], id[Uclosure[x]], E] ==
          composite[complement[inverse[S]], id[x], E]

In[90]:= composite[complement[inverse[S]], id[Uclosure[x_]], E] :=
          composite[complement[inverse[S]], id[x], E]

In[91]:= composite[inverse[E], id[Uclosure[x]], complement[S]] // DoubleInverse

Out[91]= composite[inverse[E], id[Uclosure[x]], complement[S]] ==
          composite[inverse[E], id[x], complement[S]]

In[92]:= composite[inverse[E], id[Uclosure[x_]], complement[S]] :=
          composite[inverse[E], id[x], complement[S]]

In[93]:= Assoc[composite[complement[inverse[E]], id[Uclosure[x]]], E, complement[inverse[E]]]

Out[93]= composite[complement[inverse[E]], id[Uclosure[x]], complement[inverse[S]]] ==
          composite[complement[inverse[E]], id[x], complement[inverse[S]]]

In[94]:= composite[complement[inverse[E]], id[Uclosure[x_]], complement[inverse[S]]] :=
          composite[complement[inverse[E]], id[x], complement[inverse[S]]]

In[95]:= composite[complement[S], id[Uclosure[x]], complement[E]] // DoubleInverse

Out[95]= composite[complement[S], id[Uclosure[x]], complement[E]] ==
          composite[complement[S], id[x], complement[E]]
```

```
In[96]:= composite[complement[S], id[Uclosure[x_]], complement[E]] :=
          composite[complement[S], id[x], complement[E]]

In[97]:= Assoc[composite[complement[S], id[Uclosure[x]]], complement[E], inverse[E]]

Out[97]= composite[complement[S], id[Uclosure[x]], complement[S]] ==
          composite[complement[S], id[x], complement[S]]

In[98]:= composite[complement[S], id[Uclosure[x_]], complement[S]] :=
          composite[complement[S], id[x], complement[S]]

In[99]:= composite[complement[inverse[S]],
            id[Uclosure[x]], complement[inverse[S]]] // DoubleInverse

Out[99]= composite[complement[inverse[S]], id[Uclosure[x]], complement[inverse[S]]] ==
          composite[complement[inverse[S]], id[x], complement[inverse[S]]]

In[100]:=
        composite[complement[inverse[S]], id[Uclosure[x_]], complement[inverse[S]]] :=
         composite[complement[inverse[S]], id[x], complement[inverse[S]]]
```