characterizing T1

Johan G. F. Belinfante
2014 February 8

```
In[1]:= SetDirectory["l:"]; << goedel.14feb03a
```

```
:Package Title: goedel.14feb03a                2014 February 2 at 6:15 p.m.

Loading takes about seventeen minutes, half that time due to builtin pauses.

It is now:  2014 Feb 8 at 11:13

Loading Simplification Rules

TOOLS.M is now incorporated in the GOEDEL program as of 2010 September 3

weightlimit = 40

Loading completed.

It is now:  2014 Feb 8 at 11:30
```

summary

Various characterizations of the **T1** separation condition are derived in this notebook. The results obtained are not confined to topology.

T1 separation condition

By definition, the class **T1** is the class of all sets that satisfy the following separation condition.

```
In[2]:= assert[forall[x, y, implies[and[member[x, U[t]], not[equal[x, y]]],
                    exists[z, and[member[x, z], member[z, t], not[member[y, z]]]]]]]
```

```
Out[2]= subclass[image[SINGLETON, U[t]], fix[HULL[t]]]
```

Observation. If **t** is a set, this condition is equivalent to **t ∈ T1**.

```
In[3]:= equiv[member[t, T1], and[member[t, V], subclass[image[SINGLETON, U[t]], fix[HULL[t]]]]]
```

```
Out[3]= True
```

One can replace **fix[HULL[t]]** with **Aclosure[t]** since these are equal when **t** is a set.

```
In[4]:= equiv[member[t, T1], and[member[t, V], subclass[image[SINGLETON, U[t]], Aclosure[t]]]]
```

```
Out[4]= True
```

When this definition of the class **T1** was first introduced into the **GOEDEL** program in 2003, the following different (but equivalent) expression was obtained.

```
In[5]:= equiv[member[t, T1], and[member[t, V], subclass[composite[Di, id[U[t]]],
          composite[complement[inverse[E]], HULL[t], SINGLETON]]]]

Out[5]= True
```

This equivalence also holds when **t** is not a set. This is easily established using **AssertTest**.

Theorem.

```
In[6]:= Map[implies[subclass[image[SINGLETON, U[t]], fix[HULL[t]]], #] &,
         subclass[composite[Di, id[U[t]]],
           composite[complement[inverse[E]], HULL[t], SINGLETON]] // AssertTest]

Out[6]= or[not[subclass[image[SINGLETON, U[t]], fix[HULL[t]]]],
          subclass[composite[Di, id[U[t]]],
           composite[complement[inverse[E]], HULL[t], SINGLETON]]] == True

In[7]:= or[not[subclass[image[SINGLETON, U[t_]], fix[HULL[t_]]]],
          subclass[composite[Di, id[U[t_]]],
           composite[complement[inverse[E]], HULL[t_], SINGLETON]]] := True
```

Converse Theorem.

```
In[8]:= Map[implies[#, subclass[image[SINGLETON, U[t]], fix[HULL[t]]]] &,
         subclass[composite[Di, id[U[t]]],
           composite[complement[inverse[E]], HULL[t], SINGLETON]] // AssertTest]

Out[8]= or[not[subclass[composite[Di, id[U[t]]],
            composite[complement[inverse[E]], HULL[t], SINGLETON]]],
          subclass[image[SINGLETON, U[t]], fix[HULL[t]]]] == True

In[9]:= or[not[subclass[composite[Di, id[U[t_]]],
            composite[complement[inverse[E]], HULL[t_], SINGLETON]]],
          subclass[image[SINGLETON, U[t_]], fix[HULL[t_]]]] := True
```

another characterization

From a logical point of view, it does not matter if one interchanges the order of the universal quantifications on the two variables **x** and **y** in the **T1** separation condition, but a different expression is obtained because the order of application of rewrite rules changes:

```
In[10]:= assert[forall[y, x, implies[and[member[x, U[t]], not[equal[x, y]]],
                    exists[z, and[member[x, z], member[z, t], not[member[y, z]]]]]]]

Out[10]= subclass[composite[inverse[SINGLETON], inverse[HULL[t]], E], Id]
```

Lemma. A simplification rule.

```
In[11]:= SubstTest[subclass, inverse[w], Id,
          w -> composite[inverse[SINGLETON], inverse[HULL[t]], E]]

Out[11]= subclass[composite[inverse[SINGLETON], inverse[HULL[t]], E], Id] ==
          subclass[composite[inverse[E], HULL[t], SINGLETON], Id]

In[12]:= subclass[composite[inverse[SINGLETON], inverse[HULL[t_]], E], Id] :=
          subclass[composite[inverse[E], HULL[t], SINGLETON], Id]
```

Theorem.

```
In[13]:= Map[implies[subclass[image[SINGLETON, U[t]], fix[HULL[t]]], empty[#]] &,
           dif[composite[Di, id[U[t]]], composite[complement[inverse[E]],
              HULL[t], SINGLETON]] // FastReifNormality] // Reverse

Out[13]= or[not[subclass[image[SINGLETON, U[t]], fix[HULL[t]]]],
           subclass[composite[inverse[E], HULL[t], SINGLETON], Id]] == True

In[14]:= or[not[subclass[image[SINGLETON, U[t_]], fix[HULL[t_]]]],
           subclass[composite[inverse[E], HULL[t_], SINGLETON], Id]] := True
```

Converse Theorem.

```
In[15]:= Map[implies[empty[#], subclass[image[SINGLETON, U[t]], fix[HULL[t]]]] &,
           dif[composite[Di, id[U[t]]], composite[complement[inverse[E]],
              HULL[t], SINGLETON]] // FastReifNormality] // Reverse

Out[15]= or[not[subclass[composite[inverse[E], HULL[t], SINGLETON], Id]],
           subclass[image[SINGLETON, U[t]], fix[HULL[t]]]] == True

In[16]:= or[not[subclass[composite[inverse[E], HULL[t_], SINGLETON], Id]],
           subclass[image[SINGLETON, U[t_]], fix[HULL[t_]]]] := True
```

Corollary.

```
In[17]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
            {p1 → member[t, T1], p2 -> subclass[image[SINGLETON, U[t]], fix[HULL[t]]],
             p3 -> subclass[composite[inverse[E], HULL[t], SINGLETON], Id]}]] // Reverse

Out[17]= or[not[member[t, T1]], subclass[composite[inverse[E], HULL[t], SINGLETON], Id]] == True

In[18]:= or[not[member[t_, T1]],
           subclass[composite[inverse[E], HULL[t_], SINGLETON], Id]] := True
```

Converse.

```
In[19]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[and[p0, p2], p3], not[implies[and[p0, p1], p3]],
           {p0 → member[t, v], p1 → subclass[composite[inverse[E], HULL[t], SINGLETON], Id],
            p2 -> subclass[image[SINGLETON, U[t]], fix[HULL[t]]],
            p3 -> member[t, T1]}]] // Reverse

Out[19]= or[member[t, T1], not[member[t, v]],
           not[subclass[composite[inverse[E], HULL[t], SINGLETON], Id]]] == True

In[20]:= or[member[t_, T1], not[member[t_, v_]],
           not[subclass[composite[inverse[E], HULL[t_], SINGLETON], Id]]] := True
```

Comment. In general one has:

```
In[21]:= QUASIORDER[composite[inverse[E], HULL[x], SINGLETON]]

Out[21]= True
```

The **T1** condition says that this quasi-order is trivial.

---

re-introducing variables

Lemma.

```
In[22]:= Map[or[#, not[member[y, hull[t, set[x]]]]] &,
           SubstTest[implies, and[member[u, v], subclass[v, w]],
            member[u, w], {u → pair[x, y], v -> composite[Di, id[U[t]]],
             w -> composite[complement[inverse[E]], HULL[t], SINGLETON]}]] // Reverse

Out[22]= or[equal[x, y], not[member[x, U[t]]],
           not[member[y, hull[t, set[x]]]], not[subclass[composite[Di, id[U[t]]],
             composite[complement[inverse[E]], HULL[t], SINGLETON]]]] == True

In[23]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem.

```
In[24]:= Map[not, SubstTest[and, implies[p1, p3],
           implies[and[p2, p3], p4], not[implies[and[p1, p2], p4]],
           {p1 → member[t, T1], p2 → and[member[x, U[t]], not[equal[x, y]]], p3 -> subclass[
              composite[Di, id[U[t]]], composite[complement[inverse[E]], HULL[t], SINGLETON]],
            p4 -> not[member[y, hull[t, set[x]]]]}]] // Reverse

Out[24]= or[equal[x, y], not[member[t, T1]],
           not[member[x, U[t]]], not[member[y, hull[t, set[x]]]]] == True

In[25]:= or[equal[x_, y_], not[member[t_, T1]],
           not[member[x_, U[t_]]], not[member[y_, hull[t_, set[x_]]]]] := True
```

One can eliminate the variable **y** using **reify** and **case**.

Lemma.

```
In[26]:= Map[equal[V, domain[#]] &,
          SubstTest[reify, y, case[or[equal[x, y], not[member[t, w]], not[member[x, U[t]]],
             not[member[y, hull[t, set[x]]]]]], w → T1]]

Out[26]= or[not[member[t, T1]], not[member[x, U[t]]], subclass[hull[t, set[x]], set[x]]] == True

In[27]:= (% /. {x → x_, t → t_}) /. Equal → SetDelayed
```

The result in the lemma can be improved, replacing the inclusion in the conclusion with an equation. One can do this using **AssertTest**, but the class **T1** needs to be sequestered to avoid rewriting the hypothesis **t ∈ T1**.

Theorem.

```
In[28]:= (or[not[member[t, w]], not[member[x, U[t]]],
             equal[hull[t, set[x]], set[x]]] // AssertTest) /. w → T1

Out[28]= or[equal[hull[t, set[x]], set[x]], not[member[t, T1]], not[member[x, U[t]]]] == True

In[29]:= or[equal[hull[t_, set[x_]], set[x_]],
            not[member[t_, T1]], not[member[x_, U[t_]]]] := True
```

LB[INVAR]

Several rewrite rules for **T1** derived in 2003 involved the relation **LB[INVAR]**. For example, one of the normalization rules for **T1** is this:

```
In[30]:= fix[composite[inverse[BIGCUP], E, FIRST, id[Di], inverse[SINGLETON], LB[INVAR]]]

Out[30]= complement[T1]
```

Observation. The membership rule for the relation **LB[INVAR]** is:

```
In[31]:= member[pair[x, y], LB[INVAR]]

Out[31]= and[member[x, V], member[y, V], subclass[x, invar[y]]]
```

Observation. The special case that **y** is a singleton of an ordered pair is of interest here.

```
In[32]:= class[pair[u, v], subclass[x, invar[cart[set[u], set[v]]]]]

Out[32]= union[cart[complement[U[x]], V], composite[inverse[E], HULL[x], SINGLETON]]
```

Theorem.

```
In[33]:= SubstTest[subclass, x, image[inverse[SINGLETON], t],
          {t -> image[complement[inverse[INVAR]], y]}]

Out[33]= equal[0, intersection[image[SINGLETON, x], lb[INVAR, y]]] ==
          subclass[x, composite[complement[inverse[E]], HULL[y], SINGLETON]]
```

```
In[34]:= equal[0, intersection[image[SINGLETON, x_], lb[INVAR, y_]]] :=
          subclass[x, composite[complement[inverse[E]], HULL[y], SINGLETON]]
```

a variant formulation of the T1 condition

If one adds to the **T1** condition the hypothesis that **y** belongs to **U[t]**, one obtains the following variant formulation of the **T1** condition:

```
In[35]:= assert[forall[x, y, implies[and[member[x, U[t]], member[y, U[t]], not[equal[x, y]]],
                    exists[z, and[member[x, z], member[z, t], not[member[y, z]]]]]]]
Out[35]= subclass[image[SINGLETON, U[t]], fix[HULL[t]]]
```

Lemma. A temporary rewrite rule.

```
In[36]:= composite[inverse[HULL[t]], inverse[IMAGE[id[U[t]]]]] // DoubleInverse
Out[36]= composite[inverse[HULL[t]], inverse[IMAGE[id[U[t]]]]] == inverse[HULL[t]]

In[37]:= composite[inverse[HULL[t_]], inverse[IMAGE[id[U[t_]]]]] := inverse[HULL[t]]
```

Observation. Reversing the order of the quantifiers on **x** and **y** yields:

```
In[38]:= assert[forall[y, x, implies[and[member[x, U[t]], member[y, U[t]], not[equal[x, y]]],
                    exists[z, and[member[x, z], member[z, t], not[member[y, z]]]]]]]
Out[38]= subclass[composite[inverse[E], HULL[t], SINGLETON], Id]
```

Lemma.

```
In[39]:= Assoc[IMAGE[id[complement[U[t]]]], IMAGE[id[U[t]]], HULL[t]]
Out[39]= composite[IMAGE[id[complement[U[t]]]], HULL[t]] == cart[image[inverse[S], t], set[0]]

In[40]:= composite[IMAGE[id[complement[U[t_]]]], HULL[t_]] := cart[image[inverse[S], t], set[0]]
```

Lemma. Temporary simplification rule.

```
In[41]:= ImageComp[IMAGE[id[complement[U[t]]]], HULL[t], range[SINGLETON]] // Reverse
Out[41]= image[IMAGE[id[complement[U[t]]]], image[HULL[t], range[SINGLETON]]] ==
          intersection[image[V, intersection[t, complement[set[0]]]], set[0]]

In[42]:= image[IMAGE[id[complement[U[t_]]]], image[HULL[t_], range[SINGLETON]]] :=
          intersection[image[V, intersection[t, complement[set[0]]]], set[0]]
```

Theorem. Simplification rule.

```
In[43]:= subclass[composite[inverse[E], HULL[t], SINGLETON], id[U[t]]] // AssertTest

Out[43]= subclass[composite[inverse[E], HULL[t], SINGLETON], id[U[t]]] ==
          subclass[composite[inverse[E], HULL[t], SINGLETON], Id]

In[44]:= subclass[composite[inverse[E], HULL[t_], SINGLETON], id[U[t_]]] :=
          subclass[composite[inverse[E], HULL[t], SINGLETON], Id]
```

Theorem. Simplification rule.

```
In[45]:= SubstTest[subclass, complement[u], complement[v],
          {u -> composite[complement[inverse[E]], HULL[t], SINGLETON],
           v -> composite[id[U[t]], Di, id[U[t]]]}]

Out[45]= subclass[composite[id[U[t]], Di, id[U[t]]],
           composite[complement[inverse[E]], HULL[t], SINGLETON]] ==
          subclass[composite[inverse[E], HULL[t], SINGLETON], Id]

In[46]:= subclass[composite[id[U[t_]], Di, id[U[t_]]],
           composite[complement[inverse[E]], HULL[t_], SINGLETON]] :=
          subclass[composite[inverse[E], HULL[t], SINGLETON], Id]
```

replacement rules

In this section replacements are derived for two normalization rules derived in 2003 that are related to the variant formulation of the **T1** condition.

Lemma.

```
In[47]:= member[setpart[x], T1] // AssertTest // Reverse

Out[47]= subclass[image[SINGLETON, U[setpart[x]]], Aclosure[setpart[x]]] ==
          member[setpart[x], T1]

In[48]:= subclass[image[SINGLETON, U[setpart[x_]]], Aclosure[setpart[x_]]] :=
          member[setpart[x], T1]
```

Lemma.

```
In[49]:= SubstTest[implies,
           and[member[t, V], subclass[composite[inverse[E], HULL[t], SINGLETON], Id]],
           member[t, T1], t → setpart[x]] // Reverse

Out[49]= or[member[setpart[x], T1],
           not[subclass[composite[inverse[E], HULL[setpart[x]], SINGLETON], Id]]] == True

In[50]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. Simplification rule.

```
In[51]:= equiv[subclass[composite[inverse[E], HULL[setpart[x]], SINGLETON], Id],
           member[setpart[x], T1]]

Out[51]= True

In[52]:= subclass[composite[inverse[E], HULL[setpart[x]], SINGLETON], Id] :=
           member[setpart[x], T1]
```

Theorem. Replacement rewrite rule.

```
In[53]:= Map[domain[reify[x, case[#]]] &,
           member[setpart[x], fix[composite[inverse[BIGCUP], inverse[DUP], inverse[CART],
               inverse[IMAGE[id[Di]]], S, id[range[SINGLETON]], LB[INVAR]]]] // AssertTest]

Out[53]= fix[composite[inverse[BIGCUP], inverse[DUP], inverse[CART],
             inverse[IMAGE[id[Di]]], S, id[range[SINGLETON]], LB[INVAR]]] == complement[T1]

In[54]:= fix[composite[inverse[BIGCUP], inverse[DUP], inverse[CART],
             inverse[IMAGE[id[Di]]], S, id[range[SINGLETON]], LB[INVAR]]] := complement[T1]
```

Corollary.

```
In[55]:= fix[composite[inverse[LB[INVAR]], id[range[SINGLETON]],
             inverse[S], IMAGE[id[Di]], CART, DUP, BIGCUP]] // InvertFixTest

Out[55]= fix[composite[inverse[LB[INVAR]], id[range[SINGLETON]],
             inverse[S], IMAGE[id[Di]], CART, DUP, BIGCUP]] == complement[T1]

In[56]:= fix[composite[inverse[LB[INVAR]], id[range[SINGLETON]],
             inverse[S], IMAGE[id[Di]], CART, DUP, BIGCUP]] := complement[T1]
```

yet another formula for T1

In this section, a formula for **T1** is derived based on the following observation.

```
In[57]:= member[x, invar[composite[id[range[SINGLETON]], inverse[S]]]]

Out[57]= and[member[x, V], subclass[image[SINGLETON, U[x]], x]]
```

Lemma.

```
In[59]:= Map[equal[V, domain[#]] &,
           SubstTest[reify, x, case[member[setpart[x], complement[symdif[T1, image[
                 inverse[ACLOSURE], invar[composite[inverse[E], IMAGE[t]]]]]]]], t → SINGLETON]]

Out[59]= and[subclass[
             image[inverse[ACLOSURE], invar[composite[id[range[SINGLETON]], inverse[S]]]], T1],
           subclass[intersection[T1, fix[ACLOSURE]],
             invar[composite[id[range[SINGLETON]], inverse[S]]]]] == True

In[60]:= % /. Equal → SetDelayed
```

Theorem. Another formula for **T1**.

```
In[62]:= SubstTest[and, subclass[u, v], subclass[v, u], {u → T1,
           v -> image[inverse[ACLOSURE], invar[composite[inverse[E], IMAGE[SINGLETON]]]]}]

Out[62]= equal[T1,
           image[inverse[ACLOSURE], invar[composite[id[range[SINGLETON]], inverse[S]]]]] == True

In[63]:= image[inverse[ACLOSURE], invar[composite[id[range[SINGLETON]], inverse[S]]]] := T1
```

Corollary.

```
In[64]:= ImageComp[ACLOSURE, inverse[ACLOSURE],
          invar[composite[id[range[SINGLETON]], inverse[S]]]]

Out[64]= intersection[fix[ACLOSURE], invar[composite[id[range[SINGLETON]], inverse[S]]]] ==
          intersection[T1, fix[ACLOSURE]]

In[65]:= intersection[fix[ACLOSURE], invar[composite[id[range[SINGLETON]], inverse[S]]]] :=
          intersection[T1, fix[ACLOSURE]]
```