PointClosed

Johan G. F. Belinfante
2003 December 30

```
In[1]:= << goedel52.t38; << tools.m

        :Package Title: goedel52.t38      2003 December 28 at 11:25 a.m.

        It is now:  2004 Jan 30 at 8:22

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 November 15

        weightlimit = 40
```

introduction

The class **PointClosed** captures the idea that "points are closed" without assuming that the collection is a topology. The membership rule for this class has been wrapped in **class** to make it easier to formulate theorems about it. The proofs in this notebook to some extent repeat results obtained 2003 July 3. Connections with **Uclosure** and the **T1** separation condition are explored.

characterizing the class PointClosed

Although the use of **AssertTest** has limited value in dealing with **PointClosed**, one can use this tool to derive clauses that characterize membership in the class **PointClosed**, and which can be used to reason about this class.

```
In[2]:= implies[member[x, PointClosed],
          subclass[image[SINGLETON, U[x]], image[RC[U[x]], x]]] // AssertTest

Out[2]= or[not[member[x, PointClosed]],
          subclass[image[SINGLETON, U[x]], image[RC[U[x]], x]]] == True

In[3]:= or[not[member[x_, PointClosed]],
          subclass[image[SINGLETON, U[x_]], image[RC[U[x_]], x_]]] := True

In[4]:= implies[and[member[x, y], subclass[image[SINGLETON, U[x]], image[RC[U[x]], x]]],
          member[x, PointClosed]] // AssertTest

Out[4]= or[member[x, PointClosed], not[member[x, y]],
          not[subclass[image[SINGLETON, U[x]], image[RC[U[x]], x]]]] == True

In[5]:= or[member[x_, PointClosed], not[member[x_, y_]],
          not[subclass[image[SINGLETON, U[x_]], image[RC[U[x_]], x_]]]] := True
```

complements of points are open

A second variable can be introduced into the characterization of **PointClosed**.

```
In[6]:= Map[or[#, member[dif[U[t], singleton[x]], t]] &,
          SubstTest[implies, and[member[x, u], subclass[u, v]], member[x, v],
            {u -> U[t], v -> image[inverse[SINGLETON], image[RC[U[t]], t]]}]]

Out[6]= or[member[intersection[complement[singleton[x]], U[t]], t], not[member[x, U[t]]],
          not[subclass[image[SINGLETON, U[t]], image[RC[U[t]], t]]]] == True

In[7]:= (% /. {x -> x_, t -> t_}) /. Equal -> SetDelayed
```

One can now derive a clause asserting that if **t** is in **PointClosed**, and if **x** is a point of the space **U[t]**, then the relative complement of the singleton of **x** is open.

```
In[8]:= Map[not, SubstTest[and, implies[p2, p3], implies[and[p1, p3], p4],
          not[implies[and[p1, p2], p4]], {p1 -> member[x, U[t]], p2 -> member[t, PointClosed],
           p3 -> subclass[image[SINGLETON, U[t]], image[RC[U[t]], t]],
           p4 -> member[dif[U[t], singleton[x]], t]}]]

Out[8]= or[member[intersection[complement[singleton[x]], U[t]], t],
          not[member[t, PointClosed]], not[member[x, U[t]]]] == True

In[9]:= or[member[intersection[complement[singleton[x_]], U[t_]], t_],
          not[member[t_, PointClosed]], not[member[x_, U[t_]]]] := True
```

PointClosed implies T1

First, a generally useful lemma is derived. If a point **x** belongs to an open set **y** contained in **z**, then **x** is an interior point of **z**.

```
In[10]:= SubstTest[implies, and[member[x, y], member[y, w]],
           member[x, U[w]], w -> intersection[t, P[z]]]

Out[10]= or[member[x, U[intersection[t, P[z]]]],
           not[member[x, y]], not[member[y, t]], not[subclass[y, z]]] == True

In[11]:= or[member[x_, U[intersection[t_, P[z_]]]],
           not[member[x_, y_]], not[member[y_, t_]], not[subclass[y_, z_]]] := True
```

In particular, this can be applied to the case of an interior point of the complement of a singleton:

```
In[12]:= SubstTest[implies, and[member[y, z], member[z, t], subclass[z, w]],
           member[y, U[intersection[t, P[w]]]], w -> complement[singleton[x]]]

Out[12]= or[member[x, z], member[y, U[intersection[t, P[complement[singleton[x]]]]]],
           not[member[y, z]], not[member[z, t]]] == True

In[13]:= (% /. {x -> x_, y -> y_, z -> z_, t -> t_}) /. Equal -> SetDelayed
```

Specializing to the case that the open set is the complement of a singleton yields

```
In[14]:= SubstTest[implies, and[member[y, z], member[z, t], not[member[x, z]]],
           member[y, U[intersection[t, P[complement[singleton[x]]]]]],
           z -> dif[U[t], singleton[x]]]

Out[14]= or[equal[x, y],
           member[y, U[intersection[t, P[complement[singleton[x]]]]]], not[member[y, U[t]]],
           not[member[intersection[complement[singleton[x]], U[t]], t]]] == True

In[15]:= (% /. {x -> x_, y -> y_, t -> t_}) /. Equal -> SetDelayed
```

Combining these results, one deduces that for two distinct points of the space **U[t]** associated with a point−closed collection **t**, each is an interior point of the complement of the singleton of the other:

```
In[16]:= Map[not, SubstTest[and, implies[and[p1, p2], p5], implies[and[p3, p4, p5], p6],
           not[implies[and[p1, p2, p3, p4], p6]], {p1 -> member[t, PointClosed],
            p2 -> member[x, U[t]], p3 -> member[y, U[t]], p4 -> not[equal[x, y]],
             p5 -> member[dif[U[t], singleton[x]], t],
            p6 -> member[y, U[intersection[t, P[complement[singleton[x]]]]]]}]]

Out[16]= or[equal[x, y], member[y, U[intersection[t, P[complement[singleton[x]]]]]],
           not[member[t, PointClosed]], not[member[x, U[t]]], not[member[y, U[t]]]] == True

In[17]:= (% /. {x -> x_, y -> y_, t -> t_}) /. Equal -> SetDelayed
```

The variables **x** and **y** are now removed:

```
In[18]:= Map[equal[0, composite[Id, complement[#]]] &, SubstTest[class, pair[x, y],
              or[equal[x, y],
             member[y, U[intersection[t, P[complement[singleton[x]]]]]], not[member[t, z]],
             not[member[x, U[t]]], not[member[y, U[t]]], z -> PointClosed]] // Reverse

Out[18]= or[not[member[t, PointClosed]], subclass[composite[id[U[t]], Di, id[U[t]]],
            composite[complement[inverse[E]], id[t], E]]] == True

In[19]:= (% /. t -> t_) /. Equal -> SetDelayed
```

The variable **t** is also removed:

```
In[20]:= Map[equal[V, #] &,
           SubstTest[class, t, implies[member[t, z], subclass[composite[id[U[t]], Di, id[U[t]]],
              composite[complement[inverse[E]], id[t], E]]], z -> PointClosed]] // Reverse

Out[20]= subclass[PointClosed, T1] == True

In[21]:= subclass[PointClosed, T1] := True
```

examples and counterexamples

Some examples and counterexamples of **PointClosed** collections are investigated in this section.

```
In[22]:= member[0, PointClosed] // AssertTest

Out[22]= member[0, PointClosed] == True

In[23]:= member[0, PointClosed] := True
```

Corollaries:

```
In[24]:= equal[image[S, PointClosed], V]

Out[24]= True

In[25]:= image[S, PointClosed] := V
```

```
In[26]:= Map[not, SubstTest[implies, and[member[x, y], subclass[y, z]],
           member[x, z], {x -> 0, y -> PointClosed, z -> fix[UCLOSURE]}]]

Out[26]= subclass[PointClosed, fix[UCLOSURE]] == False

In[27]:= subclass[PointClosed, fix[UCLOSURE]] := False
```

The following rule can be made more precise, but this will do for the moment:

```
In[28]:= member[singleton[0], PointClosed] // AssertTest

Out[28]= member[singleton[0], PointClosed] == True

In[29]:= % /. Equal -> SetDelayed

In[30]:= member[singleton[singleton[0]], PointClosed] // AssertTest

Out[30]= member[singleton[singleton[0]], PointClosed] == False

In[31]:= member[singleton[singleton[0]], PointClosed] := False
```

It follows that **PointClosed** is a proper subclass of **T1**.

```
In[32]:= Map[not, SubstTest[implies, and[member[x, y], subclass[y, z]],
           member[x, z], {x -> singleton[singleton[0]], y -> T1, z -> PointClosed}]]

Out[32]= subclass[T1, PointClosed] == False

In[33]:= subclass[T1, PointClosed] := False
```

the discrete topology

The discrete topology is point−closed.

```
In[34]:= member[P[x], PointClosed] // AssertTest

Out[34]= member[P[x], PointClosed] == member[x, V]

In[35]:= member[P[x_], PointClosed] := member[x, V]
```

The following lemmas are needed to derive a simple corollary from this fact.

```
In[36]:= subclass[inverse[POWER], BIGCUP] // AssertTest

Out[36]= subclass[inverse[POWER], BIGCUP] == True

In[37]:= subclass[inverse[POWER], BIGCUP] := True

In[38]:= SubstTest[implies, subclass[u, v], subclass[image[u, w], image[v, w]],
          {u -> inverse[POWER], v -> BIGCUP, w -> PointClosed}]

Out[38]= subclass[V, image[BIGCUP, PointClosed]] == True

In[39]:= % /. Equal -> SetDelayed
```

This rewrte rule amounts to the statement that there exists a point−closed topology on any set.

```
In[40]:= equal[V, image[BIGCUP, PointClosed]] // AssertTest

Out[40]= equal[V, image[BIGCUP, PointClosed]] == True

In[41]:= image[BIGCUP, PointClosed] := V
```

COARSER rule for PointClosed

Substitution lemmas:

```
In[42]:= SubstTest[implies, equal[u, v], equal[image[u, z], image[v, z]], {u -> RC[x], v -> RC[y]}]

Out[42]= or[equal[image[RC[x], z], image[RC[y], z]],
           not[equal[singleton[x], singleton[y]]]] == True

In[43]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed

In[44]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
                 {p1 -> equal[x, y], p2 -> equal[singleton[x], singleton[y]],
              p3 -> equal[image[RC[x], z], image[RC[y], z]]}]]

Out[44]= or[equal[image[RC[x], z], image[RC[y], z]], not[equal[x, y]]] == True

In[45]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed
```

The main lemma.

```
In[46]:= Map[not, SubstTest[and, implies[p1, p4], implies[p1, p5],
             implies[and[p4, p7], p8], implies[p1, p9],
           implies[and[p8, p9], p10], implies[and[p5, p10], p11],
             not[implies[and[p1, p7], p11]], {p1 -> member[pair[x, y], COARSER],
             p4 -> equal[image[SINGLETON, U[x]], image[SINGLETON, U[y]]],
             p5 -> equal[image[RC[U[x]], y], image[RC[U[y]], y]],
             p7 -> subclass[image[SINGLETON, U[x]], image[RC[U[x]], x]],
             p8 -> subclass[image[SINGLETON, U[y]], image[RC[U[x]], x]],
             p9 -> subclass[image[RC[U[x]], x], image[RC[U[x]], y]],
             p10 -> subclass[image[SINGLETON, U[y]], image[RC[U[x]], y]],
             p11 -> subclass[image[SINGLETON, U[y]], image[RC[U[y]], y]]}]]

Out[46]= or[not[equal[U[x], U[y]]], not[member[y, V]], not[subclass[x, y]],
           not[subclass[image[SINGLETON, U[x]], image[RC[U[x]], x]]],
           subclass[image[SINGLETON, U[y]], image[RC[U[y]], y]]] == True

In[47]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed
```

Theorem: If **x** is coarser than **y** and if **x** belongs to **PointClosed**, then so does **y**.

```
In[48]:= Map[not, SubstTest[and, implies[p2, p7], implies[and[p1, p11], p3],
         implies[and[p1, p7], p11],
                 not[implies[and[p1, p2], p3]], {p1 -> member[pair[x, y], COARSER],
            p2 -> member[x, PointClosed], p3 -> member[y, PointClosed],
             p7 -> subclass[image[SINGLETON, U[x]], image[RC[U[x]], x]],
             p11 -> subclass[image[SINGLETON, U[y]], image[RC[U[y]], y]]}]]

Out[48]= or[member[y, PointClosed], not[equal[U[x], U[y]]],
           not[member[x, PointClosed]], not[member[y, V]], not[subclass[x, y]]] == True
```

```
In[49]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed
```

Variable−free version:

```
In[50]:= Map[equal[0, composite[Id, complement[#]]] &, SubstTest[class, pair[x, y],
           implies[and[member[pair[x, y], w], member[x, z]], member[y, z]],
            {w -> COARSER, z -> PointClosed}]] // Reverse

Out[50]= subclass[image[COARSER, PointClosed], PointClosed] == True

In[51]:= % /. Equal -> SetDelayed
```

The reverse inclusion also holds:

```
In[52]:= SubstTest[implies, subclass[u, v],
          subclass[image[u, w], image[v, w]], {u -> Id, v -> COARSER, w -> x}]

Out[52]= subclass[x, image[COARSER, x]] == True

In[53]:= subclass[x_, image[COARSER, x_]] := True
```

Consequently, one obtains an equation:

```
In[54]:= equal[image[COARSER, PointClosed], PointClosed] // AssertTest

Out[54]= equal[PointClosed, image[COARSER, PointClosed]] == True

In[55]:= image[COARSER, PointClosed] := PointClosed
```

PointClosed is invariant under Uclosure

```
In[56]:= Map[equal[0, #] &, dif[UCLOSURE, COARSER] // VSNormality]

Out[56]= subclass[UCLOSURE, COARSER] == True

In[57]:= subclass[UCLOSURE, COARSER] := True

In[58]:= SubstTest[implies, subclass[u, v], subclass[image[u, w], image[v, w]],
            {u -> UCLOSURE, v -> COARSER, w -> PointClosed}]

Out[58]= subclass[image[UCLOSURE, PointClosed], PointClosed] == True

In[59]:= subclass[image[UCLOSURE, PointClosed], PointClosed] := True
```

T1 + Uclosed implies PointClosed

This section repeats some arguments made in 2003 July 3. Even though the topology axioms are not assumed to hold, it is easiest to motivate the proof by reference to the case of a topology. Suppose that **p** is a point of a topological space with a topology **t** that satisfies the **T1** condition. The inverse of the **T1** condition is used as follows:

```
In[60]:= SubstTest[implies, subclass[u, v],
          subclass[image[inverse[u], w], image[inverse[v], w]],
            {u -> composite[Di, id[U[t]]],
           v -> composite[complement[inverse[E]], id[t], E], w -> singleton[p]}]

Out[60]= or[not[member[p, V]],
           not[subclass[composite[Di, id[U[t]]], composite[complement[inverse[E]], id[t], E]]],
           subclass[U[t],
            union[singleton[p], U[intersection[t, P[complement[singleton[p]]]]]]]] == True
```

This result is added as a temporary simplification rule.

```
In[61]:= (% /. {p -> p_, t -> t_}) /. Equal -> SetDelayed
```

Lemma:

```
In[62]:= equal[intersection[t, P[intersection[complement[singleton[p]], U[t]]]],
           intersection[t, P[complement[singleton[p]]]]] // AssertTest

Out[62]= equal[intersection[t, P[complement[singleton[p]]]],
           intersection[t, P[intersection[complement[singleton[p]], U[t]]]]] == True

In[63]:= intersection[t_, P[intersection[complement[singleton[p_]], U[t_]]]] :=
          intersection[t, P[complement[singleton[p]]]]
```

Lemma:

```
In[64]:= equal[intersection[complement[singleton[p]], U[t]],
           U[intersection[t, P[complement[singleton[p]]]]]] // AssertTest

Out[64]= equal[intersection[complement[singleton[p]], U[t]],
           U[intersection[t, P[complement[singleton[p]]]]]] ==
          subclass[U[t], union[singleton[p], U[intersection[t, P[complement[singleton[p]]]]]]]

In[65]:= Map[implies[Last[%], #] &, %]

Out[65]= or[equal[intersection[complement[singleton[p]], U[t]],
            U[intersection[t, P[complement[singleton[p]]]]]], not[subclass[U[t],
             union[singleton[p], U[intersection[t, P[complement[singleton[p]]]]]]]]] == True

In[66]:= (% /. {p -> p_, t -> t_}) /. Equal -> SetDelayed
```

A key fact that will be used is that a set is open if it is its own interior. This fact is needed only for the (relative) complement of the singleton of a point **p**. This key fact is actually a theorem about **Uclosure**. If **t** is a topology, then **Uclosure[t] = t**, so the **Uclosure** operation would not need explicit mention.

```
In[67]:= SubstTest[implies,
          and[member[x, V], equal[x, U[intersection[t, P[x]]]], member[x, Uclosure[t]],
            x -> dif[U[t], singleton[p]]]

Out[67]= or[member[intersection[complement[singleton[p]], U[t]], Uclosure[t]],
           not[equal[intersection[complement[singleton[p]], U[t]],
             U[intersection[t, P[complement[singleton[p]]]]]]],
           not[member[intersection[complement[singleton[p]], U[t]], V]]] == True

In[68]:= (% /. {p -> p_, t -> t_}) /. Equal -> SetDelayed
```

The facts deduced can be combined:

```
In[69]:= Map[not, SubstTest[and, implies[p0, p1], implies[q, p2], implies[q, p5],
           implies[and[p1, p2], p3], implies[p3, p4], implies[p5, p6], implies[and[p4, p6], p7],
                not[implies[and[p0, q], p7]],
             { p0 -> member[p, U[t]], p1 -> member[p, V], q -> member[t, T1], p2 ->
             subclass[composite[Di, id[U[t]]], composite[complement[inverse[E]], id[t], E]],
                  p3 -> subclass[U[t], union[singleton[p],
               U[intersection[t, P[complement[singleton[p]]]]]]],
                  p4 -> equal[intersection[complement[singleton[p]], U[t]],
              U[intersection[t, P[complement[singleton[p]]]]]],
                  p5 -> member[t, V],
                  p6 -> member[intersection[complement[singleton[p]], U[t]], V],
                  p7 -> member[intersection[complement[singleton[p]], U[t]], Uclosure[t]]}]]

Out[69]= or[member[intersection[complement[singleton[p]], U[t]], Uclosure[t]],
           not[member[p, U[t]]], not[member[t, T1]]] == True

In[70]:= (% /. {p -> p_, t -> t_}) /. Equal -> SetDelayed
```

Thus, points are closed:

```
In[71]:= implies[and[member[p, U[t]], member[t, T1]],
           member[singleton[p], image[RC[U[t]], Uclosure[t]]]] // NotNotTest

Out[71]= or[and[member[t, V],
              member[intersection[complement[singleton[p]], U[t]], Uclosure[t]]],
           not[member[p, U[t]]], not[member[t, T1]]] == True

In[72]:= (% /. {p -> p_, t -> t_}) /. Equal -> SetDelayed
```

variable−free version

The first step is to remove the variable **p** from the theorem derived in the preceding section:

```
In[73]:= Map[equal[V, #] &, SubstTest[class, p,
            implies[and[member[p, U[t]], member[t, x]], member[singleton[p], y]],
             {x -> T1, y -> image[RC[U[t]], Uclosure[t]]}]] // Reverse

Out[73]= or[not[member[t, T1]],
           subclass[image[SINGLETON, U[t]], image[RC[U[t]], Uclosure[t]]]] == True

In[74]:= or[not[member[t_, T1]],
           subclass[image[SINGLETON, U[t_]], image[RC[U[t_]], Uclosure[t_]]]] := True
```

The clause characterizing membership in **PointClosed** is now specialized to the case of **Uclosure[t]**.

```
In[75]:= Map[or[not[member[t, v]], #] &, SubstTest[implies,
           and[member[x, V], subclass[image[SINGLETON, U[x]], image[RC[U[x]], x]]],
           member[x, PointClosed], x -> Uclosure[t]]]

Out[75]= or[member[Uclosure[t], PointClosed], not[member[t, v]],
           not[subclass[image[SINGLETON, U[t]], image[RC[U[t]], Uclosure[t]]]]] == True

In[76]:= (% /. {t -> t_, v -> v_}) /. Equal -> SetDelayed
```

The theorem can now be written more succinctly as follows:

```
In[77]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[and[p1, p2], p3], not[implies[p1, p3]], {p1 -> member[t, T1],
            p2 -> subclass[image[SINGLETON, U[t]], image[RC[U[t]], Uclosure[t]]],
            p3 -> member[Uclosure[t], PointClosed]}]]

Out[77]= or[member[Uclosure[t], PointClosed], not[member[t, T1]]] == True

In[78]:= (% /. t -> t_) /. Equal -> SetDelayed
```

The final step is to eliminate the variable **t**.

```
In[79]:= Map[equal[V, #] &, SubstTest[class, t, implies[member[t, x], member[Uclosure[t], y]],
                {x -> T1, y -> PointClosed}]] // Reverse

Out[79]= subclass[intersection[T1, fix[UCLOSURE]], PointClosed] == True
```

This is the variable−free formulation of the theorem.

```
In[80]:= subclass[intersection[T1, fix[UCLOSURE]], PointClosed] := True
```

The relation between **PointClosed** and **T1** can be written in a more succinct fashion.

```
In[81]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u -> PointClosed, v -> T1, w -> inverse[UCLOSURE]}]

Out[81]= subclass[image[inverse[UCLOSURE], PointClosed], T1] == True

In[82]:= % /. Equal -> SetDelayed

In[83]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> image[inverse[UCLOSURE], PointClosed], v -> T1}]

Out[83]= True == equal[T1, image[inverse[UCLOSURE], PointClosed]]

In[84]:= image[inverse[UCLOSURE], PointClosed] := T1
```

This result can be rewritten in a nice way by reintroducing a variable:

```
In[85]:= SubstTest[member, x, image[inverse[UCLOSURE], y], y -> PointClosed] // Reverse

Out[85]= member[Uclosure[x], PointClosed] == member[x, T1]

In[86]:= member[Uclosure[x_], PointClosed] := member[x, T1]
```

T1 and PointClosed coincide for Uclosed sets

In this section it is shown that for the case of a topology, there is no difference between **T1** and **PointClosed**.

```
In[87]:= ImageComp[UCLOSURE, inverse[UCLOSURE], PointClosed]

Out[87]= intersection[PointClosed, fix[UCLOSURE]] == intersection[T1, fix[UCLOSURE]]

In[88]:= intersection[PointClosed, fix[UCLOSURE]] := intersection[T1, fix[UCLOSURE]]
```

```
In[89]:= AssInt[PointClosed, fix[UCLOSURE], binclosed[CAP]]

Out[89]= intersection[PointClosed, TOPS] == intersection[T1, TOPS]

In[90]:= intersection[PointClosed, TOPS] := intersection[T1, TOPS]
```

singleton rule

A better singleton rule is derived in this section. This requires some technical lemmas:

```
In[91]:= equal[image[SINGLETON, x], singleton[0]] // AssertTest

Out[91]= equal[image[SINGLETON, x], singleton[0]] == False

In[92]:= (% /. x -> x_) /. Equal -> SetDelayed

In[93]:= image[inverse[IMAGE[SINGLETON]], succ[singleton[0]]] // Normality

Out[93]= image[inverse[IMAGE[SINGLETON]], succ[singleton[0]]] == singleton[0]

In[94]:= image[inverse[IMAGE[SINGLETON]], succ[singleton[0]]] := singleton[0]

In[95]:= image[inverse[SINGLETON], PointClosed] // Normality

Out[95]= image[inverse[SINGLETON], PointClosed] == singleton[0]

In[96]:= image[inverse[SINGLETON], PointClosed] := singleton[0]
```

Getting a result with a variable is a bit of a hassle.

```
In[97]:= SubstTest[subclass, x, image[inverse[SINGLETON], y], y -> singleton[0]] // Reverse

Out[97]= subclass[image[SINGLETON, x], singleton[0]] == subclass[x, 0]

In[98]:= subclass[image[SINGLETON, x_], singleton[0]] := subclass[x, 0]

In[99]:= member[singleton[x], PointClosed] // AssertTest

Out[99]= member[singleton[x], PointClosed] == or[equal[0, x], not[member[x, V]]]

In[100]:=
        member[singleton[x_], PointClosed] := or[equal[0, x], not[member[x, V]]]
```