self−duality of conditional completeness

Johan G. F. Belinfante
2010 September 16

```
In[1]:= << goedel.10sep14a

:Package Title: goedel.10sep14a             2010 September 14 at 4:45 p.m.

It is now:  2010 Sep 16 at 8:57

Loading Simplification Rules

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

weightlimit = 40
```

summary

That the property of being conditionally complete is selfdual is proved in Kelley's book on topology (see page 14).

```
In[2]:= "John L. Kelley, General Topology,
           D. Van Nostrand Co., Inc., Princeton, New Jersey, 1955."
```

Kelley's statement of this theorem is already quite general: he does not restrict the result to lattices, and his definition of a partial order assumes only transitivity. But even this assumption is not needed in the proof. In this notebook the theorem about selfdual−ity of the conditional completeness property is derived for any set.

the class of conditionally complete relations

The class **CC** of **conditionally complete relations** is defined by the following rewrite rule:

```
In[3]:= image[V, intersection[CC, set[x_]]] :=
         intersection[complement[image[V, intersection[x, complement[cart[V, V]]]]],
          complement[image[V, intersection[complement[domain[LUB[x]]],
             complement[set[0]], domain[UB[x]]]]], image[V, set[x]]]
```

Theorem. The class **CC** is a subclass of the class **P[cart[V, V]]** of all small relations.

```
In[4]:= Map[equal[V, #] &, complement[dif[CC, P[cart[V, V]]]] // Renormality]

Out[4]= subclass[U[CC], cart[V, V]] == True

In[5]:= subclass[U[CC], cart[V, V]] := True
```

Theorem. The empty relation is conditionally complete.

```
In[6]:= member[0, CC] // AssertTest

Out[6]= member[0, CC] == True

In[7]:= member[0, CC] := True
```

a wrapper for conditionally complete relations

A wrapper **cc[x]** for conditionally complete relations is defined by the following rewrite rule. (This rewrite rule was suggested by replacing the class **GROUPS** in the definition of the **gp[x]** wrapper by the class **CC** of conditionally complete relations.)

```
In[8]:= image[V, intersection[cc[x_], set[y_]]] :=
         intersection[complement[image[V, intersection[x, complement[cart[V, V]]]]], complement[
           image[V, intersection[complement[domain[LUB[x]]], complement[set[0]], domain[UB[x]]]]],
          image[V, intersection[x, set[y]]], image[V, set[x]]]
```

Theorem.

```
In[9]:= member[x, CC] // AssertTest // Reverse

Out[9]= and[member[x, V], subclass[x, cart[V, V]],
          subclass[domain[UB[x]], union[domain[LUB[x]], set[0]]]] == member[x, CC]

In[10]:= and[member[x_, V], subclass[x_, cart[V, V]],
          subclass[domain[UB[x_]], union[domain[LUB[x_]], set[0]]]] := member[x, CC]
```

Theorem. Normalization rule for **cc[x]**.

```
In[11]:= Map[fix, SubstTest[reify, y, image[V, intersection[t, set[y]]], t → cc[x]]] // Reverse

Out[11]= intersection[x, complement[image[V, intersection[x, complement[cart[V, V]]]]], complement[
           image[V, intersection[complement[domain[LUB[x]]], complement[set[0]], domain[UB[x]]]]],
          image[V, set[x]]] == cc[x]

In[12]:= intersection[x_, complement[image[V, intersection[x_, complement[cart[V, V]]]]],
          complement[image[V, intersection[complement[domain[LUB[x_]]],
             complement[set[0]], domain[UB[x_]]]]], image[V, set[x_]]] := cc[x]
```

Lemma.

```
In[13]:= equiv[or[equal[0, x], member[x, CC]], member[x, CC]]

Out[13]= True

In[14]:= or[equal[0, x_], member[x_, CC]] := member[x, CC]
```

Theorem. Wrapper removal rule.

```
In[15]:= SubstTest[equal, x, intersection[x, image[V, intersection[t, set[x]]]], t → CC] // Reverse

Out[15]= equal[x, cc[x]] == member[x, CC]

In[16]:= equal[x_, cc[x_]] := member[x, CC]
```

Theorem. The class **cc[x]** is a relation.

```
In[17]:= Map[empty, SubstTest[dif,
           intersection[x, complement[image[V, intersection[x, complement[cart[V, V]]]]], complement[
             image[V, intersection[complement[domain[LUB[x]]], complement[set[0]], domain[UB[x]]]]],
            image[v, set[x]]], cart[V, V], v → V]] // Reverse

Out[17]= subclass[cc[x], cart[V, V]] == True

In[18]:= subclass[cc[x_], cart[V, V]] := True
```

Theorem. The members of the class **CC** are conditionally complete.

```
In[19]:= SubstTest[implies, and[p, q], p, {p -> subclass[domain[UB[x]], union[domain[LUB[x]], set[0]]],
           q → member[x, P[cart[V, V]]]}] // Reverse
Out[19]= or[not[member[x, CC]], subclass[domain[UB[x]], union[domain[LUB[x]], set[0]]]] == True

In[20]:= or[not[member[x_, CC]], subclass[domain[UB[x_]], union[domain[LUB[x_]], set[0]]]] := True
```

Theorem. Wrapper introduction rule. The class **cc[x]** is conditionally complete.

```
In[21]:= SubstTest[member, intersection[x, image[V, intersection[t, set[x]]]], t, t → CC] // Reverse
Out[21]= member[cc[x], CC] == True

In[22]:= member[cc[x_], CC] := True
```

Theorem.

```
In[23]:= SubstTest[implies, member[t, CC],
           subclass[domain[UB[t]], union[domain[LUB[t]], set[0]]], t → cc[x]] // Reverse
Out[23]= subclass[domain[UB[cc[x]]], union[domain[LUB[cc[x]]], set[0]]] == True

In[24]:= subclass[domain[UB[cc[x_]]], union[domain[LUB[cc[x_]]], set[0]]] := True
```

Theorem. Sethood.

```
In[25]:= SubstTest[member, intersection[x, image[V, intersection[set[x], t]]], V, t → CC] // Reverse
Out[25]= member[cc[x], V] == True

In[26]:= member[cc[x_], V] := True
```

ub and lb rules for cc[x]

Theorem. Sethood rule for upper bounds of **cc[x]**.

```
In[27]:= SubstTest[or, empty[y], member[ub[setpart[t], y], V], t → cc[x]] // Reverse
Out[27]= or[equal[0, y], member[ub[cc[x], y], V]] == True

In[28]:= or[equal[0, y_], member[ub[cc[x_], y_], V]] := True
```

Corollary. A dual result.

```
In[29]:= SubstTest[or, empty[y], member[lb[setpart[t], y], V], t → cc[x]] // Reverse
Out[29]= or[equal[0, y], member[lb[cc[x], y], V]] == True

In[30]:= or[equal[0, y_], member[lb[cc[x_], y_], V]] := True
```

Theorem.

```
In[31]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u → domain[UB[cc[x]]], v -> union[domain[LUB[cc[x]]], set[0]]}]
Out[31]= equal[domain[UB[cc[x]]], union[domain[LUB[cc[x]]], set[0]]] == True

In[32]:= union[domain[LUB[cc[x_]]], set[0]] := domain[UB[cc[x]]]
```

the self−duality theorem

Lemma.

```
In[33]:= (SubstTest[implies, and[member[t, u], equal[u, v]], member[t, v], {t → lb[s, y],
           u -> domain[UB[s]], v -> union[domain[LUB[s]], set[0]]}] // Reverse) /. s → cc[x]

Out[33]= or[equal[0, lb[cc[x], y]], equal[0, ub[cc[x], lb[cc[x], y]]],
           member[lb[cc[x], y], domain[LUB[cc[x]]]], not[member[lb[cc[x], y], V]]] == True

In[34]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[35]:= Map[not, SubstTest[and, implies[p1, p3], implies[p1, p4],
           implies[and[p2, p3, p4], p5], implies[and[p3, p5], p6], not[implies[and[p1, p2], p6]],
           {p1 → not[equal[0, t]], p2 → not[equal[0, lb[cc[x], t]]], p3 → member[lb[cc[x], t], V],
            p4 → not[equal[0, ub[cc[x], lb[cc[x], t]]]], p5 → member[lb[cc[x], t], domain[LUB[cc[x]]]],
            p6 → not[equal[0, lub[cc[x], lb[cc[x], t]]]]}]] // Reverse

Out[35]= or[equal[0, t], equal[0, lb[cc[x], t]], not[equal[0, lub[cc[x], lb[cc[x], t]]]]] == True

In[36]:= (% /. {x → x_, t → t_}) /. Equal → SetDelayed
```

Lemma.

```
In[37]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[p2, p4],
           implies[and[p3, p4], p5], not[implies[and[p1, p2], p5]], {p1 → not[equal[0, t]],
            p2 → not[equal[0, lb[cc[x], t]]], p3 → not[equal[0, lub[cc[x], lb[cc[x], t]]]],
            p4 → equal[lub[cc[x], lb[cc[x], t]], intersection[ub[cc[x], lb[cc[x], t]], lb[cc[x], t]]],
            p5 → not[equal[0, intersection[ub[cc[x], lb[cc[x], t]], lb[cc[x], t]]]]}]] // Reverse

Out[37]= or[equal[0, t], equal[0, lb[cc[x], t]],
           not[equal[0, intersection[lb[cc[x], t], ub[cc[x], lb[cc[x], t]]]]]] == True

In[38]:= (% /. {x → x_, t → t_}) /. Equal → SetDelayed
```

Lemma.

```
In[39]:= Map[not,
           SubstTest[and, implies[p2, p4], implies[and[p1, p6], p7], not[implies[and[p1, p2, p3], p7]],
            {p1 → member[t, V], p2 → not[equal[0, t]], p3 → not[equal[0, lb[cc[x], t]]],
             p4 → equal[intersection[ub[cc[x], lb[cc[x], t]], lb[cc[x], t]], glb[cc[x], t]],
             p6 → not[equal[0, glb[cc[x], t]]], p7 → member[t, domain[GLB[cc[x]]]]}]] // Reverse

Out[39]= or[equal[0, t], equal[0, lb[cc[x], t]],
           member[t, domain[GLB[cc[x]]]], not[member[t, V]]] == True

In[40]:= (% /. {x → x_, t → t_}) /. Equal → SetDelayed
```

Lemma.

```
In[41]:= SubstTest[member, t, domain[LB[u]], u → cc[x]] // Reverse

Out[41]= member[t, image[inverse[S], range[VERTSECT[cc[x]]]]] ==
          and[member[t, V], not[equal[0, lb[cc[x], t]]]]

In[42]:= member[t_, image[inverse[S], range[VERTSECT[cc[x_]]]]] :=
          and[member[t, V], not[equal[0, lb[cc[x], t]]]]
```

Theorem.

```
In[43]:= Map[equal[V, #] &, SubstTest[class, t, implies[member[t, u], or[empty[t], member[t, v]]],
           {u -> domain[LB[cc[x]]], v -> domain[GLB[cc[x]]]}]]
```

```
Out[43]= subclass[image[inverse[S], range[VERTSECT[cc[x]]]], union[domain[GLB[cc[x]]], set[0]]] == True
```

```
In[44]:= subclass[image[inverse[S], range[VERTSECT[cc[x_]]]],
           union[domain[GLB[cc[x_]]], set[0]]] := True
```

Theorem. Removing the **cc** wrapper.

```
In[45]:= SubstTest[implies, equal[x, cc[t]],
           subclass[domain[LB[x]], union[domain[GLB[x]], set[0]]], t → x] // Reverse
```

```
Out[45]= or[not[member[x, CC]], subclass[domain[LB[x]], union[domain[GLB[x]], set[0]]]] == True
```

```
In[46]:= or[not[member[x_, CC]], subclass[domain[LB[x_]], union[domain[GLB[x_]], set[0]]]] := True
```

Theorem.

```
In[47]:= (member[inverse[t], CC] // AssertTest) /. t → cc[x]
```

```
Out[47]= member[inverse[cc[x]], CC] == True
```

```
In[48]:= member[inverse[cc[x_]], CC] := True
```

Corollary. (Removing the **cc** wrapper.)

```
In[49]:= SubstTest[implies, equal[x, cc[t]], member[inverse[x], CC], t → x] // Reverse
```

```
Out[49]= or[member[inverse[x], CC], not[member[x, CC]]] == True
```

```
In[50]:= or[member[inverse[x_], CC], not[member[x_, CC]]] := True
```

the selfduality theorem for any set

In this section the self−duality theorem is restated without reference to the class **CC**, and even dropping the hypothesis that **x** is a relation for brevity.

Theorem. Conditional completeness is self−dual for any set.

```
In[51]:= Map[implies[member[x, y], #] &, (or[not[member[t, V]], not[subclass[t, cart[V, V]]],
           not[subclass[domain[UB[t]], union[domain[LUB[t]], set[0]]]], subclass[domain[LB[t]],
            union[domain[GLB[t]], set[0]]]] // NotNotTest) /. t → composite[Id, x]] // MapNotNot
```

```
Out[51]= or[not[member[x, y]], not[subclass[domain[UB[x]], union[domain[LUB[x]], set[0]]],
           subclass[domain[LB[x]], union[domain[GLB[x]], set[0]]]] == True
```

```
In[52]:= or[not[member[x_, y_]], not[subclass[domain[UB[x_]], union[domain[LUB[x_]], set[0]]],
           subclass[domain[LB[x_]], union[domain[GLB[x_]], set[0]]]] := True
```

Corollary. Dual result.

```
In[53]:= Map[implies[member[x, y], #] &, SubstTest[or, not[member[t, V]],
           not[subclass[domain[UB[t]], union[domain[LUB[t]], set[0]]]], subclass[domain[LB[t]],
            union[domain[GLB[t]], set[0]]], t → inverse[x]]] // Reverse // MapNotNot
```

```
Out[53]= or[not[member[x, y]], not[subclass[domain[LB[x]], union[domain[GLB[x]], set[0]]],
           subclass[domain[UB[x]], union[domain[LUB[x]], set[0]]]] == True
```

```
In[54]:= or[not[member[x_, y_]], not[subclass[domain[LB[x_]], union[domain[GLB[x_]], set[0]]],
           subclass[domain[UB[x_]], union[domain[LUB[x_]], set[0]]]] := True
```

variable−free statement of selfduality

In this section a variable−free restatement of the self−duality theorem is derived.

Lemma.

```
In[55]:= Map[equal[V, #] &, SubstTest[class, x, or[member[inverse[x], t], not[member[x, t]]], t → CC]]
Out[55]= subclass[image[IMAGE[SWAP], CC], CC] == True

In[56]:= % /. Equal → SetDelayed
```

Lemma.

```
In[57]:= Map[subclass[#, CC] &, ImageComp[IMAGE[SWAP], id[P[cart[V, V]]], CC]]
Out[57]= subclass[image[INVERSE, CC], CC] == True

In[58]:= % /. Equal → SetDelayed
```

Theorem. A variable−free statement of the self−duality of the conditional completeness property.

```
In[59]:= equal[image[INVERSE, CC], CC] // AssertTest
Out[59]= equal[CC, image[INVERSE, CC]] == True

In[60]:= image[INVERSE, CC] := CC
```

Corollary.

```
In[61]:= ImageComp[IMAGE[SWAP], id[P[cart[V, V]]], CC] // Reverse
Out[61]= image[IMAGE[SWAP], CC] == CC

In[62]:= image[IMAGE[SWAP], CC] := CC
```