similarity for complete lattices

Johan G. F. Belinfante
2010 September 4

```
In[1]:= SetDirectory["l:"]; << goedel.10sep03a

        :Package Title: goedel.10sep03a            2010 September 3 at 2:45 p.m.

        It is now:  2010 Sep 4 at 16:28

        Loading Simplification Rules

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

        weightlimit = 40
```

summary

Any relation similar to a complete lattice is a complete lattice. This result is derived using a temporary wrapper for complete lattices contained in the domain of a given bijection. In the final section, a concise variable-free statement of the main theorem is obtained.

a temporary quasi-wrapper definition

The following rewrite rule serves to define the **c[x,y]** wrapper. This definition is analogous to the **gp[x]** wrapper for groups, and suffers from the same problems, which stem from the fact that the empty set is neither a group not a complete lattice. The definition of the new wrapper was obtained by simply replacing the class **GROUPS** of groups with the intersection of the class **CL** of complete lattices and the class **P[complement[id[complement[domain[oopart[y]]]]]]** of sets **x** that satisfy **fix[x] ⊂ domain[oopart[y]]**.

```
In[2]:= image[V, intersection[c[x_, y_], set[z_]]] := intersection[
          complement[image[V, intersection[complement[domain[GLB[x]]], P[fix[x]]]]],
          complement[image[V, intersection[complement[domain[oopart[y]]], fix[x]]]],
          image[V, intersection[PO, set[x]]], image[V, intersection[x, set[z]]]]
```

normalization

Theorem.

```
In[3]:= Map[fix, SubstTest[reify, z, image[V, intersection[t, set[z]]], t → c[x, y]]] // Reverse

Out[3]= intersection[x,
          complement[image[V, intersection[complement[domain[GLB[x]]], P[fix[x]]]]],
          complement[image[V, intersection[complement[domain[oopart[y]]], fix[x]]]],
          image[V, intersection[PO, set[x]]] == c[x, y]

In[4]:= intersection[
          complement[image[V, intersection[complement[domain[GLB[x_]]], P[fix[x_]]]]],
          complement[image[V, intersection[complement[domain[oopart[y_]]], fix[x_]]]],
          image[V, intersection[PO, set[x_]]], x_] := c[x, y]
```

wrapper removal and introduction rules

The wrapper removal rule is derived using **reify** in the same way that was done for the **gp[x]** wrapper. Note that **c[x, y]** is not a true wrapper because it need not be a complete lattice in general. It could also be the empty set. This minor nuisance was also encountered with the **gp[x]** wrapper, and our previous experience will repeatedly be helpful as a guide in dealing with this problem.

Theorem. (Wrapper removal rule.)

```
In[5]:= SubstTest[equal, x, intersection[x, image[V, intersection[t, set[x]]]],
          t → intersection[CL, P[complement[id[complement[domain[oopart[y]]]]]]]] // Reverse

Out[5]= equal[x, c[x, y]] ==
         or[and[member[x, CL], subclass[fix[x], domain[oopart[y]]]], equal[0, x]]

In[6]:= equal[x_, c[x_, y_]] :=
         or[and[member[x, CL], subclass[fix[x], domain[oopart[y]]]], equal[0, x]]
```

A temporary introduction rule will be derived, which will later be replaced with simpler rewrite rules. The following lemma is helpful.

Lemma.

```
In[7]:= SubstTest[member,
          intersection[x, image[V, intersection[t, set[x]]], t, t → union[set[0],
            intersection[CL, P[complement[id[complement[domain[oopart[y]]]]]]]]] // Reverse

Out[7]= or[and[member[c[x, y], CL], subclass[fix[c[x, y]], domain[oopart[y]]]],
          equal[0, c[x, y]]] == True

In[8]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. A temporary wrapper introduction rule.

```
In[9]:= equiv[and[member[c[x, y], CL], subclass[fix[c[x, y]], domain[oopart[y]]]],
         not[equal[0, c[x, y]]]]

Out[9]= True
```

```
In[10]:= and[member[c[x_, y_], CL], subclass[fix[c[x_, y_]], domain[oopart[y_]]]] :=
          not[equal[0, c[x, y]]]
```

Comment. The wrapper introduction rule often produces literals of the form **equal[0, c[x,y]]**, but these are quite easy to remove. The following simple lemma is helpful in this regard.

Theorem.

```
In[11]:= equiv[and[member[x, CL], not[equal[0, x]]], member[x, CL]]

Out[11]= True

In[12]:= and[member[x_, CL], not[equal[0, x_]]] := member[x, CL]
```

---

sethood rule

Theorem. The wrapper **c[x, y]** always denotes a set.

```
In[13]:= SubstTest[member, intersection[x, image[V, intersection[set[x], t]]], V,
           t → intersection[CL, P[complement[id[complement[domain[oopart[y]]]]]]]] // Reverse

Out[13]= member[c[x, y], V] == True

In[14]:= member[c[x_, y_], V] := True
```

PARTIALORDER rules

Many elementary properties of the wrapper **c[x, y]** follow from the fact that **c[x, y]** is always a partial order.

Lemma. (This contains a redundant literal that will be removed momentarily.)

```
In[15]:= SubstTest[implies, and[member[t, CL], subclass[fix[t], domain[oopart[y]]]],
           member[t, PO], t → c[x, y]] // Reverse

Out[15]= or[equal[0, c[x, y]], PARTIALORDER[c[x, y]]] == True

In[16]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Since **0** is also a partial order, the redundant literal is easily eliminated.

Theorem. The wrapper **c[x, y]** is always a partial order.

```
In[17]:= SubstTest[and, or[p, q], implies[p, q],
           {p -> equal[0, c[x, y]], q -> PARTIALORDER[c[x, y]]}]

Out[17]= PARTIALORDER[c[x, y]] == True

In[18]:= PARTIALORDER[c[x_, y_]] := True
```

Various useful corollaries of this fact will now be deduced.

Corollary. The class **c[x, y]** is a relation.

```
In[19]:= SubstTest[composite, Id, po[t], t → c[x, y]] // Reverse
Out[19]= composite[Id, c[x, y]] == c[x, y]

In[20]:= composite[Id, c[x_, y_]] := c[x, y]
```

Corollary. The domain of **c[x, y]** is the same as its fixed point class.

```
In[21]:= SubstTest[domain, po[t], t → c[x, y]] // Reverse
Out[21]= domain[c[x, y]] == fix[c[x, y]]

In[22]:= domain[c[x_, y_]] := fix[c[x, y]]
```

Corollary. The range of **c[x, y]** is the same as its fixed point class.

```
In[23]:= SubstTest[range, po[t], t → c[x, y]] // Reverse
Out[23]= range[c[x, y]] == fix[c[x, y]]

In[24]:= range[c[x_, y_]] := fix[c[x, y]]
```

Theorem. The class **c[x, y]** is empty if and only if its fixed point class is empty.

```
In[25]:= SubstTest[empty, fix[po[t]], t → c[x, y]] // Reverse
Out[25]= equal[0, fix[c[x, y]]] == equal[0, c[x, y]]

In[26]:= equal[0, fix[c[x_, y_]]] := equal[0, c[x, y]]
```

Corollary. A simplification rule.

```
In[27]:= image[V, fix[c[x, y]]] // Normality
Out[27]= image[V, fix[c[x, y]]] == image[V, c[x, y]]

In[28]:= image[V, fix[c[x_, y_]]] := image[V, c[x, y]]
```

Theorem. A rule for cartesian product upper bounds.

```
In[29]:= SubstTest[subclass, composite[Id, t], cart[u, v], t → c[x, y]] // Reverse
Out[29]= subclass[c[x, y], cart[u, v]] == and[subclass[fix[c[x, y]], u], subclass[fix[c[x, y]], v]]

In[30]:= subclass[c[x_, y_], cart[u_, v_]] :=
          and[subclass[fix[c[x, y]], u], subclass[fix[c[x, y]], v]]
```

domain of GLB

In this section the temporary introduction rule for **c[x, y]** is replaced with simpler rewrite rules.

Theorem. The **GLB** relation is a function.

```
In[31]:= SubstTest[FUNCTION, GLB[po[t]], t → c[x, y]] // Reverse
Out[31]= FUNCTION[GLB[c[x, y]]] == True

In[32]:= FUNCTION[GLB[c[x_, y_]]] := True
```

Lemma.

```
In[33]:= SubstTest[implies, and[member[t, CL], subclass[fix[t], domain[oopart[y]]]],
           equal[P[fix[t]], domain[GLB[t]]], t → c[x, y]] // Reverse
Out[33]= or[equal[0, c[x, y]], equal[domain[GLB[c[x, y]]], P[fix[c[x, y]]]]] == True

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

To obtain an equation for the domain of the **GLB[c[x, y]]** function, one must take into account that **c[x, y]** could be empty. This is easily accomplished by intersecting with **image[V, c[x, y]]**.

Theorem. An explicit equation for **domain[GLB[c[x, y]]]**.

```
In[35]:= equal[domain[GLB[c[x, y]]], intersection[P[fix[c[x, y]]], image[V, c[x, y]]]] //
          not // not
Out[35]= True

In[36]:= domain[GLB[c[x_, y_]]] := intersection[image[V, c[x, y]], P[fix[c[x, y]]]]
```

An explicit statement about membership in the class **CL** of complete lattices will now be derived. This result is completely analogous to the corresponding rule for the **gp[x]** wrapper.

Lemma.

```
In[37]:= or[equal[0, c[x, y]], member[c[x, y], CL]] // AssertTest
Out[37]= or[equal[0, c[x, y]], member[c[x, y], CL]] == True

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

Theorem. The class **c[x, y]** is a complete lattice if and only if it is not the empty set.

```
In[39]:= equiv[member[c[x, y], CL], not[empty[c[x, y]]]]
Out[39]= True

In[40]:= member[c[x_, y_], CL] := not[equal[0, c[x, y]]]
```

The temporary wrapper introduction rule derived earlier is transformed by this rule.

Theorem. Replacement for an earlier rule.

```
In[41]:= SubstTest[member,
           intersection[x, image[V, intersection[t, set[x]]], t, t → union[set[0],
             intersection[CL, P[complement[id[complement[domain[oopart[y]]]]]]]]] // Reverse

Out[41]= or[equal[0, c[x, y]], subclass[fix[c[x, y]], domain[oopart[y]]]] == True

In[42]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Again, one can remove a redundant literal.

Theorem. A better rewrite rule.

```
In[43]:= SubstTest[and, or[p, q], implies[p, q],
          {p -> equal[0, c[x, y]], q -> subclass[fix[c[x, y]], domain[oopart[y]]]}]

Out[43]= subclass[fix[c[x, y]], domain[oopart[y]]] == True

In[44]:= subclass[fix[c[x_, y_]], domain[oopart[y_]]] := True
```

domain of LUB

For later work it proves to be more convenient to deal with **LUB** instead of **GLB** because this avoids a nuisance rewrite rule for the lower bound relation of a set. The details are quite similar to those obtained in the preceding section.

Lemma.

```
In[45]:= SubstTest[implies, member[t, CL],
           equal[P[fix[t]], domain[LUB[t]]], t → c[x, y]] // Reverse

Out[45]= or[equal[0, c[x, y]], equal[domain[LUB[c[x, y]]], P[fix[c[x, y]]]]] == True

In[46]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. An explicit equation for **domain[LUB[c[x, y]]]**.

```
In[47]:= equal[domain[LUB[c[x, y]]], intersection[P[fix[c[x, y]]], image[V, c[x, y]]]] // not //
          not

Out[47]= True

In[48]:= domain[LUB[c[x_, y_]]] := intersection[image[V, c[x, y]], P[fix[c[x, y]]]]
```

Theorem. A general formula for the domain of the **LUB** function of a partial order.

```
In[49]:= Map[domain, SubstTest[composite, id[range[t]],
           intersection[complement[composite[complement[inverse[t]], id[range[t]], UB[t]]],
            UB[t]], t → po[x]]] // Reverse

Out[49]= fix[composite[
           complement[composite[inverse[UB[po[x]]], id[fix[po[x]]], complement[po[x]]]],
           id[fix[po[x]]], UB[po[x]]]] == domain[LUB[po[x]]]

In[50]:= fix[composite[
           complement[composite[inverse[UB[po[x_]]], id[fix[po[x_]]], complement[po[x_]]]],
           id[fix[po[x_]]], UB[po[x_]]]] := domain[LUB[po[x]]]
```

There is an analogous formula for the **GLB** function, which can be derived using duality.

Corollary.

```
In[51]:= SubstTest[fix, composite[
           complement[composite[inverse[UB[po[t]]], id[fix[po[t]]], complement[po[t]]]],
           id[fix[po[t]]], UB[po[t]]], t → inverse[x]] // Reverse

Out[51]= fix[composite[complement[
            composite[inverse[LB[po[x]]], id[fix[po[x]]], complement[inverse[po[x]]]]],
           id[fix[po[x]]], LB[po[x]]]] == domain[GLB[po[x]]]

In[52]:= fix[composite[complement[
            composite[inverse[LB[po[x_]]], id[fix[po[x_]]], complement[inverse[po[x_]]]]],
           id[fix[po[x_]]], LB[po[x_]]]] := domain[GLB[po[x]]]
```

The latter formula involves **LB**, which currently is automatically rewritten as when the argument is a set. For example:

```
In[53]:= LB[c[x, y]]

Out[53]= composite[inverse[VERTSECT[c[x, y]], S]
```

To avoid this problem, the **LUB** formula will be used exclusively for the remainder of this notebook.

Theorem. A simplification formula obtained by specializing the formula for the domain of **LUB** to the case of **c[x, y]**.

```
In[54]:= SubstTest[fix, composite[
           complement[composite[inverse[UB[po[t]]], id[fix[po[t]]], complement[po[t]]]],
           id[fix[po[t]]], UB[po[t]]], t → c[x, y]] // Reverse

Out[54]= fix[composite[
           complement[composite[inverse[UB[c[x, y]]], id[fix[c[x, y]]], complement[c[x, y]]]],
           id[fix[c[x, y]]], UB[c[x, y]]]] ==
          intersection[image[V, c[x, y]], P[fix[c[x, y]]]]

In[55]:= fix[composite[complement[composite[inverse[UB[c[x_, y_]]], id[fix[c[x_, y_]]],
              complement[c[x_, y_]]]], id[fix[c[x_, y_]]], UB[c[x_, y_]]]] :=
          intersection[image[V, c[x, y]], P[fix[c[x, y]]]]
```

simplification rules

At this point if one tries to specialize the **domain[LUB[po[t]]]** formula to the case **t = oopart[y] ∘ c[x,y] ∘ inverse[oopart[y]]** one easily encounters execution times in excess of a minute. To deal with this problem additional rewrite rules are needed.

Lemma.

```
In[56]:= SubstTest[implies, member[t, CL],
           not[equal[0, intersection[fix[t], lb[t, fix[t]]]]], t → c[x, y]] // Reverse

Out[56]= or[equal[0, c[x, y]],
           not[equal[0, intersection[fix[c[x, y]], lb[c[x, y], fix[c[x, y]]]]]]] == True

In[57]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. A better rewrite rule.

```
In[58]:= equiv[equal[0, intersection[fix[c[x, y]], lb[c[x, y], fix[c[x, y]]]]], equal[0, c[x, y]]]

Out[58]= True

In[59]:= equal[0, intersection[fix[c[x_, y_]], lb[c[x_, y_], fix[c[x_, y_]]]]] :=
          equal[0, c[x, y]]
```

Corollary.

```
In[60]:= image[V, intersection[fix[c[x, y]], lb[c[x, y], fix[c[x, y]]]]] // Normality

Out[60]= image[V, intersection[fix[c[x, y]], lb[c[x, y], fix[c[x, y]]]]] == image[V, c[x, y]]

In[61]:= image[V, intersection[fix[c[x_, y_]], lb[c[x_, y_], fix[c[x_, y_]]]]] :=
          image[V, c[x, y]]
```

Theorem. A key simplification rule.

```
In[62]:= equal[intersection[fix[c[x, y]], domain[oopart[y]]], fix[c[x, y]]]

Out[62]= True

In[63]:= intersection[domain[oopart[y_]], fix[c[x_, y_]]] := fix[c[x, y]]
```

Theorem. A partial order rule.

```
In[64]:= SubstTest[PARTIALORDER,
           composite[oopart[y], po[t], inverse[oopart[y]]], t → c[x, y]] // Reverse

Out[64]= PARTIALORDER[composite[oopart[y], c[x, y], inverse[oopart[y]]]] == True

In[65]:= PARTIALORDER[composite[oopart[y_], c[x_, y_], inverse[oopart[y_]]]] := True
```

Theorem. A sethood rule.

```
In[66]:= member[composite[oopart[y], c[x, y], inverse[oopart[y]]], V] // AssertTest

Out[66]= member[composite[oopart[y], c[x, y], inverse[oopart[y]]], V] == True

In[67]:= member[composite[oopart[y_], c[x_, y_], inverse[oopart[y_]]], V] := True
```

Theorem. A criterion for **oopart[y]** ∘ **c[x,y]** ∘ **inverse[oopart[y]]** to be a complete lattice.

```
In[68]:= SubstTest[and, member[t, PO], subclass[P[fix[t]], domain[LUB[t]]],
           t -> composite[oopart[y], c[x, y], inverse[oopart[y]]]] // Reverse

Out[68]= subclass[P[image[oopart[y], fix[c[x, y]]]],
           domain[LUB[composite[oopart[y], c[x, y], inverse[oopart[y]]]]]] ==
          member[composite[oopart[y], c[x, y], inverse[oopart[y]]], CL]

In[69]:= subclass[P[image[oopart[y_], fix[c[x_, y_]]]],
           domain[LUB[composite[oopart[y_], c[x_, y_], inverse[oopart[y_]]]]]] :=
          member[composite[oopart[y], c[x, y], inverse[oopart[y]]], CL]
```

two rules involving the IMAGE function

Two rewrite rules about the **IMAGE** function were discovered that are needed for simplifying expressions.

Theorem. A transformation rule for certain **fix** expressions.

```
In[70]:= SubstTest[fix, composite[inverse[funpart[t]], y, funpart[t]],
           t -> composite[IMAGE[inverse[oopart[x]]], id[P[range[oopart[x]]]]]] // Reverse

Out[70]= intersection[fix[composite[IMAGE[oopart[x]], id[P[domain[oopart[x]]]],
             y, IMAGE[inverse[oopart[x]]]], P[range[oopart[x]]]] ==
          image[IMAGE[oopart[x]], intersection[fix[y], P[domain[oopart[x]]]]]

In[71]:= intersection[fix[composite[IMAGE[oopart[x_]], id[P[domain[oopart[x_]]]],
             y_, IMAGE[inverse[oopart[x_]]]], P[range[oopart[x_]]]] :=
          image[IMAGE[oopart[x]], intersection[fix[y], P[domain[oopart[x]]]]]
```

Theorem.

```
In[72]:= member[0, image[IMAGE[x], y]] // AssertTest

Out[72]= member[0, image[IMAGE[x], y]] ==
          not[equal[0, intersection[y, P[complement[domain[x]]]]]]

In[73]:= member[0, image[IMAGE[x_], y_]] :=
          not[equal[0, intersection[y, P[complement[domain[x]]]]]]
```

main theorem

For the main theorem, it helps to temporarily clear the **simplify** flag. Doing so reduces execution time from over a minute to a few seconds.

```
In[74]:= simplify = False;
```

Lemma.

```
In[75]:= Map[subclass[P[image[oopart[y], fix[c[x, y]]]], #] &, SubstTest[fix, composite[
           complement[composite[inverse[UB[po[t]]], id[fix[po[t]]], complement[po[t]]]],
           id[fix[po[t]]], UB[po[t]]], t → composite[oopart[y], c[x, y], inverse[oopart[y]]]]]

Out[75]= member[composite[oopart[y], c[x, y], inverse[oopart[y]]], CL] == not[equal[0, c[x, y]]]

In[76]:= member[composite[oopart[y_], c[x_, y_], inverse[oopart[y_]]], CL] :=
          not[equal[0, c[x, y]]]
```

The main theorem is obtained by using the wrapper removal rule for **c[x, y]**.

Main theorem. A relation similar to a complete lattice order is a complete lattice order.

```
In[77]:= SubstTest[implies, equal[t, c[x, y]],
           or[equal[0, t], member[composite[oopart[y], t, inverse[oopart[y]]], CL]],
           t → x] // Reverse // MapNotNot

Out[77]= or[member[composite[oopart[y], x, inverse[oopart[y]]], CL],
           not[member[x, CL]], not[subclass[fix[x], domain[oopart[y]]]]] == True

In[78]:= or[member[composite[oopart[y_], x_, inverse[oopart[y_]]], CL],
           not[member[x_, CL]], not[subclass[fix[x_], domain[oopart[y_]]]]] := True
```

One can remove the **oopart** wrapper as well.

Corollary.

```
In[79]:= SubstTest[implies, equal[x, oopart[u]], or[member[composite[x, y, inverse[x]], CL],
           not[member[y, CL]], not[subclass[fix[y], domain[x]]]], u → x] // Reverse

Out[79]= or[member[composite[x, y, inverse[x]], CL],
           not[FUNCTION[x]], not[FUNCTION[inverse[x]]],
           not[member[y, CL]], not[subclass[fix[y], domain[x]]]] == True

In[80]:= or[member[composite[x_, y_, inverse[x_]], CL],
           not[FUNCTION[x_]], not[FUNCTION[inverse[x_]]],
           not[member[y_, CL]], not[subclass[fix[y_], domain[x_]]]] := True
```

variable-free restatement

In this section a succinct variable-free restatement of the main theorem is derived.

Lemma. A variant of the main theorem.

```
In[81]:= SubstTest[implies, equal[x, po[w]],
           or[member[y, CL], not[equal[y, composite[t, x, inverse[t]]]],
            not[FUNCTION[t]], not[FUNCTION[inverse[t]]], not[member[x, CL]],
            not[subclass[x, cart[domain[t], domain[t]]]]], w → x] // Reverse // MapNotNot

Out[81]= or[member[y, CL], not[equal[y, composite[t, x, inverse[t]]]],
           not[FUNCTION[t]], not[FUNCTION[inverse[t]]], not[member[x, CL]],
           not[subclass[x, cart[domain[t], domain[t]]]]] == True

In[82]:= or[member[y_, CL], not[equal[composite[t_, x_, inverse[t_]], y_]],
           not[FUNCTION[inverse[t_]]], not[FUNCTION[t_]], not[member[x_, CL]],
           not[subclass[x_, cart[domain[t_], domain[t_]]]]] := True
```

Lemma. (A negative form of the previous lemma, with an additional variable.)

```
In[83]:= and[equal[composite[x, y, inverse[x]], z],
           FUNCTION[inverse[x]], FUNCTION[x], member[y, CL], not[member[z, CL]],
           subclass[y, cart[domain[x], domain[x]]]] // NotNotTest

Out[83]= and[equal[z, composite[x, y, inverse[x]]], FUNCTION[x], FUNCTION[inverse[x]],
           member[y, CL], not[member[z, CL]], subclass[y, cart[domain[x], domain[x]]]] == False

In[84]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

All three variables are removed at the same time. Note the use of the **setpart** wrapper to speed up execution. The **simplify** flag needs to be reset for this.

```
In[86]:= simplify = True;
```

Lemma. The class of complete lattices is invariant under the similarity relation.

```
In[87]:= Map[empty, SubstTest[class, pair[pair[x, y], z],
           member[pair[pair[setpart[x], setpart[y]], setpart[z]], t],
           t -> composite[id[complement[CL]], IMG, cross[composite[CROSS, DUP], Id],
             id[composite[id[CL], inverse[S], CART, DUP, IMAGE[FIRST], id[BIJ]]]]]]

Out[87]= subclass[image[SIMILAR, CL], CL] == True

In[88]:= % /. Equal → SetDelayed
```

Theorem. A more concise rewrite rule.

In[89]:= **SubstTest[and, subclass[u, v], subclass[v, u], {u -> image[SIMILAR, CL], v → CL}]**

Out[89]= equal[CL, image[SIMILAR, CL]] == True

In[90]:= **image[SIMILAR, CL] := CL**