complete lattice wrapper cl[x]

Johan G. F. Belinfante
2010 September 7

```
In[1]:= SetDirectory["l:"]; << goedel.10sep04b

        :Package Title: goedel.10sep04b            2010 September 4 at 7:35 p.m.

        It is now:  2010 Sep 7 at 6:39

        Loading Simplification Rules

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

        weightlimit = 40
```

summary

A wrapper **cl[x]** for complete lattices is introduced by analogy with wrapper **gp[x]** for groups. Both wrappers suffer from the nuisance that although the empty set is neither a group nor a complete lattice, each of the sets **cl[x]** and **gp[x]** could be empty.

definition of cl[x]

The following rewrite rule which serves to define the **cl[x]** wrapper was suggested by simply replacing the class **GROUPS** of groups with the class **CL** of complete lattices.

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

normalization

The following normalization condition for **cl[x]** is derived in the same way that was done for **gp[x]**.

Theorem.

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

Out[3]= intersection[x,
          complement[image[V, intersection[complement[domain[GLB[x]]], P[fix[x]]]]],
          image[V, intersection[PO, set[x]]]] == cl[x]
```

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

wrapper removal and introduction rules

The derivation of the wrapper removal rule is derived using **reify** just like the corresponding rule for the **gp[x]** wrapper. Note that **cl[x]** is not a true wrapper as 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 this previous experience is often 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 → CL] // Reverse

Out[5]= equal[x, cl[x]] == or[equal[0, x], member[x, CL]]

In[6]:= equal[x_, cl[x_]] := or[equal[0, x], member[x, CL]]
```

An introduction rule will be derived using following lemma.

Lemma.

```
In[7]:= SubstTest[member, intersection[x, image[V, intersection[t, set[x]]]],
          t, t → union[set[0], CL]] // Reverse

Out[7]= or[equal[0, cl[x]], member[cl[x], CL]] == True

In[8]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. Wrapper introduction rule. The class **cl[x]** is a complete lattice if and only if it is not the empty set.

```
In[9]:= equiv[member[cl[x], CL], not[equal[0, cl[x]]]]

Out[9]= True

In[10]:= member[cl[x_], CL] := not[equal[0, cl[x]]]
```

Comment. The wrapper introduction rule often produces literals of the form **equal[0, cl[x]]**, but these are usually easy to remove.

Corollary. Idempotence of the **cl** wrapper.

```
In[11]:= equal[cl[cl[x]], cl[x]]

Out[11]= True

In[12]:= cl[cl[x_]] := cl[x]
```

sethood rule

Theorem. The wrapper **cl[x]** always denotes a set.

```
In[13]:= SubstTest[member,
           intersection[x, image[V, intersection[set[x], t]]], V, t → CL] // Reverse

Out[13]= member[cl[x], V] == True

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

partial order rules

Many elementary properties of the wrapper **cl[x]** follow from the fact that this set is always a partial order.

Theorem. The set **cl[x]** is a partial order.

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

Out[15]= PARTIALORDER[cl[x]] == True

In[16]:= PARTIALORDER[cl[x_]] := True
```

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

```
In[17]:= SubstTest[composite, Id, po[t], t → cl[x]] // Reverse

Out[17]= composite[Id, cl[x]] == cl[x]

In[18]:= composite[Id, cl[x_]] := cl[x]
```

Theorem. A rule for cartesian product upper bounds.

```
In[19]:= SubstTest[subclass, po[t], cart[y, z], t → cl[x]] // Reverse

Out[19]= subclass[cl[x], cart[y, z]] == and[subclass[fix[cl[x]], y], subclass[fix[cl[x]], z]]

In[20]:= subclass[cl[x_], cart[y_, z_]] := and[subclass[fix[cl[x]], y], subclass[fix[cl[x]], z]]
```

Theorem. The relation **cl[x]** is idempotent.

```
In[21]:= SubstTest[idempotent, po[t], t → cl[x]] // Reverse

Out[21]= equal[cl[x], composite[cl[x], cl[x]]] == True

In[22]:= composite[cl[x_], cl[x_]] := cl[x]
```

Corollary. The relation **cl[x]** is antisymmetric.

```
In[23]:= SubstTest[intersection, po[t], inverse[po[t]], t → cl[x]] // Reverse
Out[23]= intersection[cl[x], inverse[cl[x]]] == id[fix[cl[x]]]

In[24]:= intersection[cl[x_], inverse[cl[x_]]] := id[fix[cl[x]]]
```

Theorem. The domain of **cl[x]** is the same as its fixed point class.

```
In[25]:= SubstTest[domain, po[t], t → cl[x]] // Reverse
Out[25]= domain[cl[x]] == fix[cl[x]]

In[26]:= domain[cl[x_]] := fix[cl[x]]
```

Theorem. The range of **cl[x]** is the same as its fixed point class.

```
In[27]:= SubstTest[range, po[t], t → cl[x]] // Reverse
Out[27]= range[cl[x]] == fix[cl[x]]

In[28]:= range[cl[x_]] := fix[cl[x]]
```

Theorem. The set **cl[x]** is empty if and only if its fixed point class is empty.

```
In[29]:= SubstTest[empty, fix[po[t]], t → cl[x]] // Reverse
Out[29]= equal[0, fix[cl[x]]] == equal[0, cl[x]]

In[30]:= equal[0, fix[cl[x_]]] := equal[0, cl[x]]
```

Corollary. A simplification rule.

```
In[31]:= image[V, fix[cl[x]]] // Normality
Out[31]= image[V, fix[cl[x]]] == image[V, cl[x]]

In[32]:= image[V, fix[cl[x_]]] := image[V, cl[x]]
```

Theorem. A rule for the inverse of **cl[x]** useful for duality proofs.

```
In[33]:= equal[cl[inverse[cl[x]]], inverse[cl[x]]]
Out[33]= True

In[34]:= cl[inverse[cl[x_]]] := inverse[cl[x]]
```

domain of GLB and LUB

Observation. Since **cl[x]** is antisymmetric, its **GLB** relation is a function. No new rewrite rule is needed.

```
In[35]:= FUNCTION[GLB[cl[x]]]
```

```
Out[35]= True
```

Lemma.

```
In[36]:= SubstTest[implies, member[t, CL], equal[P[fix[t]], domain[GLB[t]]], t → cl[x]] // Reverse
```

```
Out[36]= or[equal[0, cl[x]], equal[domain[GLB[cl[x]]], P[fix[cl[x]]]]] == True
```

```
In[37]:= (% /. x → x_) /. Equal → SetDelayed
```

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

Theorem. An explicit equation for **domain[GLB[cl[x]]]**.

```
In[38]:= equal[domain[GLB[cl[x]]], intersection[P[fix[cl[x]]], image[V, cl[x]]]] // not // not
```

```
Out[38]= True
```

```
In[39]:= domain[GLB[cl[x_]]] := intersection[image[V, cl[x]], P[fix[cl[x]]]]
```

Corollary. Dual result.

```
In[40]:= SubstTest[domain, GLB[cl[t]], t → inverse[cl[x]]] // Reverse
```

```
Out[40]= domain[LUB[cl[x]]] == intersection[image[V, cl[x]], P[fix[cl[x]]]]
```

```
In[41]:= domain[LUB[cl[x_]]] := intersection[image[V, fix[cl[x]]], P[fix[cl[x]]]]
```

reify rule

Theorem.

```
In[42]:= SubstTest[reify, x,
           intersection[f[x], image[V, intersection[t, set[f[x]]]]], t → CL] // Reverse
```

```
Out[42]= reify[x, cl[f[x]]] ==
          composite[reify[x, f[x]], id[image[inverse[VERTSECT[reify[x, f[x]]], CL]]]
```

```
In[43]:= reify[x_, cl[y_]] :=
          composite[reify[x, y], id[image[inverse[VERTSECT[reify[x, y]]], CL]]]
```

The **reify** rule is useful for eliminatimh variables. Here are two simple examples.

Theorem.

```
In[44]:= Map[composite[VERTSECT[#], id[CL]] &, SubstTest[reify, x, domain[f[x]], f → cl]]
```

```
Out[44]= composite[IMAGE[FIRST], id[CL]] == composite[IMAGE[inverse[DUP]], id[CL]]
```

```
In[45]:= composite[IMAGE[FIRST], id[CL]] := composite[IMAGE[inverse[DUP]], id[CL]]
```

Theorem.

```
In[46]:= Map[composite[VERTSECT[#], id[CL]] &, SubstTest[reify, x, range[f[x]], f → cl]]
Out[46]= composite[IMAGE[SECOND], id[CL]] == composite[IMAGE[inverse[DUP]], id[CL]]
```

```
In[47]:= composite[IMAGE[SECOND], id[CL]] := composite[IMAGE[inverse[DUP]], id[CL]]
```

---

domain of UB

In this section, a formula for the domain of the upper bound relation of a complete lattice is derived.

Lemma.

```
In[48]:= SubstTest[subclass, domain[LUB[t]], domain[UB[t]], t → cl[x]] // Reverse
Out[48]= or[equal[0, cl[x]], subclass[P[fix[cl[x]]], domain[UB[cl[x]]]]] == True
```

```
In[49]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[50]:= SubstTest[and, or[p, q], implies[p, q],
          {p -> equal[0, cl[x]], q -> subclass[P[fix[cl[x]]], domain[UB[cl[x]]]]}]
Out[50]= subclass[P[fix[cl[x]]], domain[UB[cl[x]]]] == True
```

```
In[51]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[52]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> P[fix[cl[x]]], v -> domain[UB[cl[x]]]}]
Out[52]= equal[domain[UB[cl[x]]], P[fix[cl[x]]]] == True
```

```
In[53]:= domain[UB[cl[x_]]] := P[fix[cl[x]]]
```

Theorem. A variable-free restatement obtained using **reify**.

```
In[54]:= Map[composite[VERTSECT[#], id[CL]] &, SubstTest[reify, x, domain[UB[f[x]]], f → cl]]
Out[54]= composite[UBD, id[CL]] == composite[POWER, IMAGE[inverse[DUP]], id[CL]]
```

```
In[55]:= composite[UBD, id[CL]] := composite[POWER, IMAGE[inverse[DUP]], id[CL]]
```