oopart of a total order

Johan G. F. Belinfante
2009 May 25

```
In[1]:= SetDirectory["l:"]; << goedel.09may24a; << tools.m

        :Package Title: goedel.09may24a                 2009 May 24 at 10:20 a.m.

        It is now:  2009 May 25 at 11:39

        Loading Simplification Rules

        TOOLS.M                                    Revised 2009 May 16

        weightlimit = 40
```

summary

The oopart (one-to-one part) of a total order is its restriction to the set of elements that are simultaneously minimal and maximal.

```
In[2]:= intersection[minimal[to[x], fix[to[x]]], maximal[to[x], fix[to[x]]]]

Out[2]= fix[oopart[to[x]]]
```

In this notebook it is shown that unless a total order consists of a single point, then its oopart is empty. The idea is this. If the oopart is not empty, then it consists of a single point **pair[u,u]**, where **u** is both minimal and maximal. If the total order were not a singleton, then there would be another point, say **v**, not equal to **u**, in its fixed point class. By dichotomy, **pair[u,v]** must either belong to the total order or its inverse. In either case, since **u** is both minimal and maximal, one must have **u = v**. This is a contradiction. A converse holds. If a total order does consist of a single point, then its oopart is itself, and hence is not empty. Combining the theorem and its converse yields a rewrite rule that transforms the statement that the oopart of a total order is empty into the statement that the total order is not a singleton.

derivation

Lemma. An element **u** of **fix[oopart[to[x]]]** is maximal. That is, there is no element **v** greater than **u**.

```
In[3]:= (Map[implies[#, equal[u, v]] &, SubstTest[member, pair[u, v],
             composite[t, id[y]], y → fix[oopart[t]]]] /. t → to[x]) // MapNotNot

Out[3]= or[equal[u, v], not[member[u, fix[oopart[to[x]]]]],
          not[member[pair[u, v], to[x]]]] == True

In[4]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Lemma. An element **v** of **fix[oopart[to[x]]]** is minimal. That is, there is no element **u** less than **v**.

```
In[5]:= (Map[implies[#, equal[u, v]] &, SubstTest[member, pair[u, v],
           composite[id[y], t], y → fix[oopart[t]]]] /. t → to[x]) // MapNotNot

Out[5]= or[equal[u, v], not[member[v, fix[oopart[to[x]]]]],
          not[member[pair[u, v], to[x]]] == True

In[6]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

In a total order, any two elements are related. So, if an element is both minimal and maximal, then there is no other element.

Theorem. If an element **u** belongs to **fix[oopart[to[x]]]**, then **u** is the only member of **fix[to[x]]**.

```
In[7]:= Map[not, SubstTest[and, implies[and[p1, p2, p3], p4], implies[and[p1, p2, p3], p5],
           implies[and[p4, p5], p6], implies[and[p1, p2], not[p6]], and[p1, p2, p3],
           {p1 → member[u, fix[oopart[to[x]]]], p2 → member[v, fix[to[x]]], p3 → not[equal[u, v]],
            p4 → not[member[pair[u, v], to[x]]], p5 -> not[member[pair[v, u], to[x]]],
            p6 → not[member[pair[u, v], cartsq[fix[to[x]]]]]}]] // Reverse

Out[7]= or[equal[u, v], not[member[u, fix[oopart[to[x]]]]], not[member[v, fix[to[x]]]]] == True

In[8]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Eliminating the variables **u** and **v** yields the following.

Corollary. Either a total order is empty, or its oopart is empty, or the fixed point classes of the total order and its oopart are equal.

```
In[9]:= Map[empty[composite[Id, complement[#]]] &,
         SubstTest[class, pair[u, v], or[equal[u, v], not[member[u, y]], not[member[v, z]]],
          {y -> fix[oopart[to[x]]], z -> fix[to[x]]}]]

Out[9]= or[equal[0, oopart[to[x]]],
          equal[0, to[x]], equal[fix[oopart[to[x]]], fix[to[x]]]] == True

In[10]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. If the fixed point class of a total order and its oopart are the same, then the total order is either empty or a singleton.

```
In[11]:= SubstTest[implies, and[equal[u, v], member[v, w]], member[u, w],
           {v → fix[oopart[to[x]]], u → fix[to[x]], w → range[SINGLETON]}] // Reverse

Out[11]= or[equal[0, oopart[to[x]]], member[to[x], range[SINGLETON]],
           not[equal[fix[oopart[to[x]]], fix[to[x]]]] == True

In[12]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. If a total order is not a singleton, then its oopart is empty.

```
In[13]:= Map[not, SubstTest[and, implies[p1, or[p3, p4]],
           implies[and[p1, p4], p2], implies[and[p1, p3], p2], not[implies[p1, p2]],
           {p1 → not[empty[oopart[to[x]]]], p2 → member[to[x], range[SINGLETON]],
            p3 → empty[to[x]], p4 -> equal[fix[oopart[to[x]]], fix[to[x]]]}]] // Reverse

Out[13]= or[equal[0, oopart[to[x]]], member[to[x], range[SINGLETON]]] == True

In[14]:= (% /. x → x_) /. Equal→ SetDelayed
```

---

removing a rewrite rule

The following rewrite rule, which was used above, will soon be replaced by a more direct one.

```
In[15]:= equal[0, fix[oopart[to[x]]]]

Out[15]= equal[0, oopart[to[x]]]
```

This rule will now be removed:

```
In[16]:= equal[0, fix[oopart[to[x_]]]] =.
```

---

converse

Lemma. If a total order is a singleton, then it is the identity relation restriced to a single point.

```
In[17]:= Map[implies[member[to[x], #], subclass[to[x], Id]] &,
           AssInt[TO, RFX, range[SINGLETON]]] // Reverse

Out[17]= or[not[member[to[x], range[SINGLETON]]], subclass[to[x], Id]] == True

In[18]:= (% /. x → x_) /. Equal→ SetDelayed
```

Lemma. The oopart of a singleton identity relation is not empty.

```
In[19]:= SubstTest[implies, and[equal[x, id[t]], member[x, image[SINGLETON, Id]]],
           not[empty[oopart[x]]], t → fix[x]] // Reverse

Out[19]= or[not[equal[0, oopart[x]]],
           not[member[x, range[SINGLETON]]], not[subclass[x, Id]]] == True

In[20]:= (% /. x → x_) /. Equal→ SetDelayed
```

Theorem. If a total order is a singleton, then its oopart is not empty.

```
In[21]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3],
            not[implies[p1, p3]], {p1 → member[to[x], range[SINGLETON]],
             p2 -> subclass[to[x], Id], p3 -> not[equal[0, oopart[to[x]]]]}]] // Reverse

Out[21]= or[not[equal[0, oopart[to[x]]]], not[member[to[x], range[SINGLETON]]]] == True
```

```
In[22]:= (% /. x → x_) /. Equal → SetDelayed
```

Combining the theorem of the preceding section with this converse yields a logical equivalence that can be made into a rewrite rule.

Theorem.

```
In[23]:= equiv[empty[oopart[to[x]]], not[member[to[x], range[SINGLETON]]]]
Out[23]= True

In[24]:= equal[0, oopart[to[x_]]] := not[member[to[x], range[SINGLETON]]]
```

replacement for the removed rule

The removed rewrite rule can now be replaced by a more direct rule.

```
In[25]:= SubstTest[empty, fix[oopart[po[t]]], t → to[x]] // Reverse
Out[25]= equal[0, fix[oopart[to[x]]]] == not[member[to[x], range[SINGLETON]]]

In[26]:= equal[0, fix[oopart[to[x_]]]] := not[member[to[x], range[SINGLETON]]]
```