a partial order wrapper

Johan G. F. Belinfante
2005 October 1

```
In[1]:= SetDirectory["l:"]; << goedel73.27c; << tools.m

:Package Title: goedel73.27c            2005 September 27 at 4:55 p.m.

It is now:  2005 Oct 1 at 7:47

Loading Simplification Rules

TOOLS.M                      Revised 2005 September 30

weightlimit = 40
```

summary

A wrapper for partial order relations is introduced, and a few of its properties are derived.

definition of po[x] and some basic properties of this wrapper

The definition of **po[x]** has been carefully crafted to make it easy to derive new result using duality.

```
In[2]:= member[w_, po[x_]] :=
         and[member[w, x], member[first[w], V], PARTIALORDER[composite[Id, x]]]
```

It is immediate from the definition that **po[x]** is a relation.

```
In[3]:= Map[equal[V, #] &,
          SubstTest[class, y, implies[member[y, u], member[y, cart[V, V]]], u → po[x]]] // Reverse

Out[3]= subclass[po[x], cart[V, V]] == True

In[4]:= subclass[po[x_], cart[V, V]] := True
```

Corollary.

```
In[5]:= equal[composite[Id, po[x]], po[x]]

Out[5]= True

In[6]:= composite[Id, po[x_]] := po[x]
```

The **po** wrapper ignores ordered pairs.

```
In[7]:= equal[po[composite[Id, x]], po[x]] // AssertTest

Out[7]= equal[po[x], po[composite[Id, x]]] == True

In[8]:= po[composite[Id, x_]] := po[x]
```

Another immediate consequence of the definition is this inclusion:

```
In[9]:= subclass[po[x], x] // AssertTest

Out[9]= subclass[po[x], x] == True

In[10]:= subclass[po[x_], x_] := True
```

To speed up the derivation of the two main properties of the **po** wrapper, the **simplify** flags are cleared temporarily.

```
In[11]:= simplify = False;

In[12]:= PARTIALORDER[po[x]] // AssertTest

Out[12]= PARTIALORDER[po[x]] == True

In[13]:= PARTIALORDER[po[x_]] := True

In[14]:= equal[x, po[x]] // AssertTest

Out[14]= equal[x, po[x]] == PARTIALORDER[x]

In[15]:= equal[x_, po[x_]] := PARTIALORDER[x]
```

Adding a conditional rewrite rule as well is useful:

```
In[16]:= po[x_] := x /; PARTIALORDER[x]
```

Corollary.

```
In[17]:= SubstTest[equal, y, po[y], y → composite[Id, x]]

Out[17]= equal[composite[Id, x], po[x]] == PARTIALORDER[composite[Id, x]]

In[18]:= equal[composite[Id, x_], po[x_]] := PARTIALORDER[composite[Id, x]]
```

The duality property is derived as follows:

```
In[19]:= equal[inverse[po[x]], po[inverse[x]]] // AssertTest

Out[19]= equal[inverse[po[x]], po[inverse[x]]] == True

In[20]:= po[inverse[x_]] := inverse[po[x]]
```

The above rule is oriented by analogy with similar rewrite rules for the closely related wrappers **rfx** and **trv**.

REFLEXIVE property

The reflexive property of **po[x]** is studied in the section. The basic result is obtained as follows:

```
In[21]:= SubstTest[implies, PARTIALORDER[y], REFLEXIVE[y], y → po[x]]

Out[21]= REFLEXIVE[po[x]] == True

In[22]:= REFLEXIVE[po[x_]] := True
```

Many corollaries follow from this. Any statement involving the **rfx** wrapper can be converted to a corresponding one for **po** by using this fact:

```
In[23]:= rfx[po[x]]

Out[23]= po[x]
```

Here is a simple example:

```
In[24]:= SubstTest[domain, rfx[y], y → po[x]]

Out[24]= domain[po[x]] == fix[po[x]]

In[25]:= domain[po[x_]] := fix[po[x]]
```

The same technique obviously will work also for **range**, but here we choose instead to illustrate the use of duality.

```
In[26]:= SubstTest[domain, po[y], y → inverse[x]]

Out[26]= range[po[x]] == fix[po[x]]

In[27]:= range[po[x_]] := fix[po[x]]
```

ANTISYMMETRIC property

The **ANTISYMMETRIC** property breaks up into two parts, one of which has already been dealt with:

```
In[28]:= ANTISYMMETRIC[x]

Out[28]= and[subclass[x, cart[V, V]], subclass[intersection[x, inverse[x]], Id]]
```

For the other part, one has the following result, which we add as a temporary rewrite rule:

```
In[29]:= SubstTest[implies, PARTIALORDER[y],
          subclass[intersection[y, inverse[y]], Id], y → po[x]]

Out[29]= subclass[intersection[inverse[po[x]], po[x]], Id] == True

In[30]:= (% /. x → x_) /. Equal → SetDelayed
```

This temporary rule is subsumed by the following corollary, which serves better as a permanent rewrite rule:

```
In[31]:= equal[intersection[inverse[po[x]], po[x]], id[fix[po[x]]]]

Out[31]= True

In[32]:= intersection[inverse[po[x_]], po[x_]] := id[fix[po[x]]]
```

Various other corollaries are now automatic and do not require separate rewrite rules:

```
In[33]:= FUNCTION[GLB[po[x]]]

Out[33]= True

In[34]:= FUNCTION[LUB[po[x]]]

Out[34]= True
```

Any result involving the **rfx** wrapper can be used to derive a corollary for **po** because of the following fact:

```
In[35]:= rfx[po[x]]

Out[35]= po[x]
```

TRANSITIVE property

The basic rule about transitivity is this:

```
In[36]:= SubstTest[implies, PARTIALORDER[y], TRANSITIVE[y], y → po[x]]

Out[36]= TRANSITIVE[po[x]] == True

In[37]:= TRANSITIVE[po[x_]] := True
```

Many corollaries follow from this. Any statement involving the **trv** wrapper can be converted to a corresponding one for **po** by using this fact:

```
In[38]:= trv[po[x]]

Out[38]= po[x]
```

In particular, any rewrite rule conerning reflexive transitive relations can be converted to a corresponding result about partial orders. For example:

```
In[39]:= SubstTest[composite, rfx[trv[y]], rfx[trv[y]], y → po[x]]

Out[39]= composite[po[x], po[x]] == po[x]

In[40]:= composite[po[x_], po[x_]] := po[x]
```

Another result about reflexive transitive relations involves composites with the inverse complement:

```
In[41]:= SubstTest[composite, complement[inverse[rfx[trv[y]]]], rfx[trv[y]], y → po[x]]

Out[41]= composite[complement[inverse[po[x]]], po[x]] ==
          composite[complement[inverse[po[x]]], id[fix[po[x]]]]

In[42]:= composite[complement[inverse[po[x_]]], po[x_]] :=
          composite[complement[inverse[po[x]]], id[fix[po[x]]]]
```

There are in all four such formulas. The others can also be derived in exactly the same way, or as an alternative, one can make use of duality:

```
In[43]:= SubstTest[composite, complement[inverse[po[y]]], po[y], y → inverse[x]]

Out[43]= composite[complement[po[x]], inverse[po[x]]] ==
          composite[complement[po[x]], id[fix[po[x]]]]

In[44]:= composite[complement[po[x_]], inverse[po[x_]]] :=
          composite[complement[po[x]], id[fix[po[x]]]]
```

The remaining two such rules can be derived using **DoubleInverse**.

```
In[45]:= composite[po[x], complement[inverse[po[x]]]] // DoubleInverse

Out[45]= composite[po[x], complement[inverse[po[x]]]] ==
          composite[id[fix[po[x]]], complement[inverse[po[x]]]]

In[46]:= composite[po[x_], complement[inverse[po[x_]]]] :=
          composite[id[fix[po[x]]], complement[inverse[po[x]]]]

In[47]:= composite[inverse[po[x]], complement[po[x]]] // DoubleInverse

Out[47]= composite[inverse[po[x]], complement[po[x]]] ==
          composite[id[fix[po[x]]], complement[po[x]]]

In[48]:= composite[inverse[po[x_]], complement[po[x_]]] :=
          composite[id[fix[po[x]]], complement[po[x]]]
```