a partial order wrapper

Johan G. F. Belinfante
2005 October 1

In[1]:= SetDirectory["1:"]; << 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 \[Rule] 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.
Another immediate consequence of the definition is this inclusion:

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

Adding a conditional rewrite rule as well is useful:

Corollary.

The duality property is derived as follows:

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 \( \text{po}[x] \) is studied in the section. The basic result is obtained as follows:

\[
\text{In} [21] := \text{SubstTest}[\text{implies, PARTIALORDER}[y], \text{REFLEXIVE}[y], y \rightarrow \text{po}[x]]
\]

\[
\text{Out} [21] = \text{REFLEXIVE}[\text{po}[x]] = \text{True}
\]

\[
\text{In} [22] := \text{REFLEXIVE}[\text{po}[x_\_] ] := \text{True}
\]

Many corollaries follow from this. Any statement involving the \( \text{rfx} \) wrapper can be converted to a corresponding one for \( \text{po} \) by using this fact:

\[
\text{In} [23] := \text{rfx}[\text{po}[x]]
\]

\[
\text{Out} [23] = \text{po}[x]
\]

Here is a simple example:

\[
\text{In} [24] := \text{SubstTest}[\text{domain, rfx}[y], y \rightarrow \text{po}[x]]
\]

\[
\text{Out} [24] = \text{domain}[\text{po}[x]] = \text{fix}[\text{po}[x]]
\]

\[
\text{In} [25] := \text{domain}[\text{po}[x_\_] ] := \text{fix}[\text{po}[x]]
\]

The same technique obviously will work also for \( \text{range} \), but here we choose instead to illustrate the use of duality.

\[
\text{In} [26] := \text{SubstTest}[\text{domain, po}[y], y \rightarrow \text{inverse}[x]]
\]

\[
\text{Out} [26] = \text{range}[\text{po}[x]] = \text{fix}[\text{po}[x]]
\]

\[
\text{In} [27] := \text{range}[\text{po}[x_\_] ] := \text{fix}[\text{po}[x]]
\]

**ANTISYMMETRIC property**

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

\[
\text{In} [28] := \text{ANTISYMMETRIC}[x]
\]

\[
\text{Out} [28] = \text{and}[\text{subclass}[x, \text{cart}[V, V]], \text{subclass}[\text{intersection}[x, \text{inverse}[x]], \text{Id}]]
\]

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

\[
\text{In} [29] := \text{SubstTest}[\text{implies, PARTIALORDER}[y],\text{subclass}[\text{intersection}[y, \text{inverse}[y]], \text{Id}], y \rightarrow \text{po}[x]]
\]

\[
\text{Out} [29] = \text{subclass}[\text{intersection}[\text{inverse}[\text{po}[x]], \text{po}[x]], \text{Id}] = \text{True}
\]

\[
\text{In} [30] := (\% / . x \rightarrow x_\_) / . \text{Equal} \rightarrow \text{SetDelayed}
\]
This temporary rule is subsumed by the following corollary, which serves better as a permanent rewrite rule:

\[
\text{In } 31: = \text{equal[intersection[inverse[po[x]], po[x]], id[fix[po[x]]]]}
\]
\[
\text{Out } 31: = \text{True}
\]

\[
\text{In } 32: = \text{intersection[inverse[po[x]], po[x]] := id[fix[po[x]]]}
\]

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

\[
\text{In } 33: = \text{FUNCTION[GLB[po[x]]]}
\]
\[
\text{Out } 33: = \text{True}
\]

\[
\text{In } 34: = \text{FUNCTION[LUB[po[x]]]}
\]
\[
\text{Out } 34: = \text{True}
\]

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

\[
\text{In } 35: = \text{rfx[po[x]]}
\]
\[
\text{Out } 35: = \text{po[x]}
\]

---

**TRANSITIVE property**

The basic rule about transitivity is this:

\[
\text{In } 36: = \text{SubstTest[implies, PARTIALORDER[y], TRANSITIVE[y], y \rightarrow po[x]]}
\]
\[
\text{Out } 36: = \text{TRANSITIVE[po[x]]} = \text{True}
\]

\[
\text{In } 37: = \text{TRANSITIVE[po[x]]} := \text{True}
\]

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

\[
\text{In } 38: = \text{trv[po[x]]}
\]
\[
\text{Out } 38: = \text{po[x]}
\]

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

\[
\text{In } 39: = \text{SubstTest[composite, rfx[trv[y]], rfx[trv[y]], y \rightarrow po[x]]}
\]
\[
\text{Out } 39: = \text{composite[po[x], po[x]]} = \text{po[x]}
\]

\[
\text{In } 40: = \text{composite[po[x], po[x]]} := \text{po[x]}
\]

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

SubstTest[composite, complement[inverse[rfx[trv[y]]]], rfx[trv[y]], y \to po[x]]

Out

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

In 

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

Out

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 

SubstTest[composite, complement[inverse[po[y]]], po[y], y \to inverse[x]]

Out

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

In 

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

Out

The remaining two such rules can be derived using DoubleInverse.

In 

composite[po[x], complement[inverse[po[x]]]] \ DoubleInverse

Out

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

In 

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

Out

DoubleInverse

composite[inverse[po[x]], complement[po[x]]] // DoubleInverse

In 

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

Out

In 

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