the predicate TOTALORDER

Johan G. F. Belinfante
2004 February 10

```
In[1]:= << goedel54.10a; << tools.m

        :Package Title: goedel54.10a      2004 February 10 at 12:20 noon

        It is now:  2004 Feb 11 at 12:18

        Loading Simplification Rules

        TOOLS.M                       Revised 2004 January 3

        weightlimit = 40
```

summary

The definition of the unary predicate **TOTALORDER** has been wrapped to facilitate reasoning:

```
In[2]:= Begin["Goedel`Private`"];

In[3]:= InfoMatch[class[w_, HoldPattern[TOTALORDER[x_]]]] // First // First

Out[3]= class[w_, TOTALORDER[x_]] := class[w, and[ANTISYMMETRIC[x],
          TRANSITIVE[x], equal[union[x, inverse[x]], cart[fix[x], fix[x]]]]]
```

Basic facts about this predicate are derived in this notebook.

basics

The **simplify** flag is turned off to speed up the process of deriving facts about the predicate **TOTALORDER**.

```
In[4]:= simplify = False;
```

One can establish some elementary facts about **TOTALORDER** by using nothing more than **AssertTest**.

```
In[5]:= implies[TOTALORDER[x], subclass[x, cart[V, V]]] // AssertTest

Out[5]= or[not[TOTALORDER[x]], subclass[x, cart[V, V]]] == True

In[6]:= or[not[TOTALORDER[x_]], subclass[x_, cart[V, V]]] := True
```

Total orderings can be characterized as reflexive antisymmetric transitive relations **x** for which any two members of **fix[x]** can be compared.

```
In[7]:= implies[TOTALORDER[x], REFLEXIVE[x]] // AssertTest

Out[7]= or[not[TOTALORDER[x]], subclass[x, cart[fix[x], fix[x]]]] == True
```

```
In[8]:= or[not[TOTALORDER[x_]], subclass[x_, cart[fix[x_], fix[x_]]]] := True

In[9]:= implies[TOTALORDER[x], equal[cart[fix[x], fix[x]], union[x, inverse[x]]]] // AssertTest

Out[9]= or[equal[cart[fix[x], fix[x]], union[x, inverse[x]]], not[TOTALORDER[x]]] == True

In[10]:= or[equal[cart[fix[x_], fix[x_]], union[x_, inverse[x_]]], not[TOTALORDER[x_]]] := True

In[11]:= implies[TOTALORDER[x], TRANSITIVE[x]] // AssertTest

Out[11]= or[not[TOTALORDER[x]], TRANSITIVE[x]] == True

In[12]:= or[not[TOTALORDER[x_]], TRANSITIVE[x_]] := True

In[13]:= implies[TOTALORDER[x], subclass[intersection[x, inverse[x]], Id]] // AssertTest

Out[13]= or[not[TOTALORDER[x]], subclass[intersection[x, inverse[x]], Id]] == True

In[14]:= or[not[TOTALORDER[x_]], subclass[intersection[x_, inverse[x_]], Id]] := True
```

The domain and range of a total order are both equal to its fixed point set.

```
In[15]:= implies[TOTALORDER[x], equal[domain[x], fix[x]]] // AssertTest

Out[15]= or[equal[domain[x], fix[x]], not[TOTALORDER[x]]] == True

In[16]:= or[equal[domain[x_], fix[x_]], not[TOTALORDER[x_]]] := True

In[17]:= implies[TOTALORDER[x], equal[range[x], fix[x]]] // AssertTest

Out[17]= or[equal[fix[x], range[x]], not[TOTALORDER[x]]] == True

In[18]:= or[equal[fix[x_], range[x_]], not[TOTALORDER[x_]]] := True
```

The basic result in the opposite direction is the following. Note that the comparability condition implies reflexivity, so that can be omitted.

```
In[19]:= or[not[equal[cart[fix[x], fix[x]], union[x, inverse[x]]]],
           not[subclass[intersection[x, inverse[x]], Id]],
           not[TRANSITIVE[x]], TOTALORDER[x]] // AssertTest

Out[19]= or[not[equal[cart[fix[x], fix[x]], union[x, inverse[x]]]],
           not[subclass[intersection[x, inverse[x]], Id]],
           not[TRANSITIVE[x]], TOTALORDER[x]] == True

In[20]:= or[not[equal[cart[fix[x_], fix[x_]], union[x_, inverse[x_]]]],
           not[subclass[intersection[x_, inverse[x_]], Id]],
           not[TRANSITIVE[x_]], TOTALORDER[x_]] := True
```

Corollary: A total order is a partial order in which any pair of elements in **fix[x]** are comparable:

```
In[21]:= implies[and[PARTIALORDER[x],
           equal[cart[fix[x], fix[x]], union[x, inverse[x]]]], TOTALORDER[x]]

Out[21]= True
```

equality substitution

The rule for equality substitution is derived in this section. The following lemma is a temporary one:

```
In[22]:= equiv[and[equal[x, y], equal[composite[Id, x], composite[Id, y]]], equal[x, y]]

Out[22]= True

In[23]:= and[equal[x_, y_], equal[composite[Id, x_], composite[Id, y_]]] := equal[x, y]
```

The following rule is needed to derive an equality substitution rule for antisymmetry:

```
In[24]:= SubstTest[implies, and[equal[u, v], equal[x, y]],
           equal[intersection[u, x], intersection[v, y]],
           {u -> inverse[x], v -> inverse[y]}] // MapNotNot

Out[24]= or[equal[intersection[x, inverse[x]], intersection[y, inverse[y]]], not[equal[x, y]]] ==
          True

In[25]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

This is the equality substitution rule for antisymmetry:

```
In[26]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[and[p2, p3], p4], not[implies[and[p1, p3], p4]], {p1 -> equal[x, y],
            p2 -> equal[intersection[x, inverse[x]], intersection[y, inverse[y]]],
                 p3 -> subclass[intersection[x, inverse[x]], Id],
            p4 -> subclass[intersection[y, inverse[y]], Id]}]]

Out[26]= or[not[equal[x, y]], not[subclass[intersection[x, inverse[x]], Id]],
           subclass[intersection[y, inverse[y]], Id]] == True

In[27]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

Lemma.

```
In[28]:= SubstTest[implies, and[equal[u, v], equal[v, z]],
          equal[u, z], {u -> cart[x, x], v -> cart[y, y]}]

Out[28]= or[equal[z, cart[x, x]], not[equal[x, y]], not[equal[z, cart[y, y]]]] == True

In[29]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed
```

Lemma.

```
In[30]:= SubstTest[implies, equal[x, y], equal[image[w, x], image[w, y]], w -> union[Id, SWAP]]

Out[30]= or[equal[union[x, inverse[x]], union[y, inverse[y]]], not[equal[x, y]]] == True

In[31]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

This is an equality substitution rule for comparability:

```
In[32]:= Map[not, SubstTest[and, implies[p2, p7], implies[p2, p8],
           implies[and[p7, p9], p10], implies[and[p8, p10], p11],
                not[implies[and[p2, p9], p11]], {p2 -> equal[x, y], p7 -> equal[fix[x], fix[y]],
            p8 -> equal[union[x, inverse[x]], union[y, inverse[y]]],
                p9 -> equal[cart[fix[x], fix[x]], union[x, inverse[x]]],
            p10 -> equal[cart[fix[y], fix[y]], union[x, inverse[x]]],
                p11 -> equal[cart[fix[y], fix[y]], union[y, inverse[y]]]}]]

Out[32]= or[equal[cart[fix[y], fix[y]], union[y, inverse[y]]], not[equal[x, y]],
           not[equal[cart[fix[x], fix[x]], union[x, inverse[x]]]]] == True

In[33]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

Double negation can be used to bundle various hypotheses:

```
In[34]:= or[and[equal[cart[fix[x], fix[x]], union[x, inverse[x]]], subclass[
             intersection[x, inverse[x]], Id], TRANSITIVE[x]], not[TOTALORDER[x]]] // NotNotTest

Out[34]= or[and[equal[cart[fix[x], fix[x]], union[x, inverse[x]]], subclass[
             intersection[x, inverse[x]], Id], TRANSITIVE[x]], not[TOTALORDER[x]]] == True

In[35]:= (% /. x -> x_) /. Equal -> SetDelayed
```

Double negation is used to bundle the various substitution rules derived above:

```
In[36]:= or[and[equal[cart[fix[y], fix[y]], union[y, inverse[y]]],
              subclass[intersection[y, inverse[y]], Id], TRANSITIVE[y]],
            not[equal[x, y]], not[equal[cart[fix[x], fix[x]], union[x, inverse[x]]]],
            not[subclass[intersection[x, inverse[x]], Id]], not[TRANSITIVE[x]]] // NotNotTest

Out[36]= or[and[equal[cart[fix[y], fix[y]], union[y, inverse[y]]],
              subclass[intersection[y, inverse[y]], Id], TRANSITIVE[y]],
            not[equal[x, y]], not[equal[cart[fix[x], fix[x]], union[x, inverse[x]]]],
            not[subclass[intersection[x, inverse[x]], Id]], not[TRANSITIVE[x]]] == True

In[37]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

This is the equality substiution rule for **TOTALORDER**.

```
In[38]:= Map[not, SubstTest[and, implies[p1, p3], implies[and[p2, p3], p4], implies[p4, p5],
           not[implies[and[p1, p2], p5]], {p1 -> TOTALORDER[x], p2 -> equal[x, y],
            p3 -> and[TRANSITIVE[x], subclass[intersection[x, inverse[x]], Id],
              equal[cart[fix[x], fix[x]], union[x, inverse[x]]]],
                 p4 -> and[TRANSITIVE[y], subclass[intersection[y, inverse[y]], Id],
              equal[cart[fix[y], fix[y]], union[y, inverse[y]]]], p5 -> TOTALORDER[y]}]]

Out[38]= or[not[equal[x, y]], not[TOTALORDER[x]], TOTALORDER[y]] == True

In[39]:= or[not[equal[x_, y_]], not[TOTALORDER[x_]], TOTALORDER[y_]] := True
```

some elementary theorems

If **x** is a total order, then **x = composite[Id, x]**, so the following holds:

```
In[40]:= implies[TOTALORDER[x], TOTALORDER[composite[Id, x]]] // AssertTest

Out[40]= or[not[TOTALORDER[x]], TOTALORDER[composite[Id, x]]] == True

In[41]:= or[not[TOTALORDER[x_]], TOTALORDER[composite[Id, x_]]] := True
```

The equality substitution theorem is useful in the other direction:

```
In[42]:= SubstTest[implies, and[equal[x, y], TOTALORDER[y]],
          TOTALORDER[x], y -> composite[Id, x]]

Out[42]= or[not[subclass[x, cart[V, V]]],
           not[TOTALORDER[composite[Id, x]]], TOTALORDER[x]] == True

In[43]:= or[not[subclass[x_, cart[V, V]]],
           not[TOTALORDER[composite[Id, x_]]], TOTALORDER[x_]] := True
```

Theorem: The inverse of a total order is a total order.

```
In[44]:= implies[TOTALORDER[x], TOTALORDER[inverse[x]]] // AssertTest

Out[44]= or[not[TOTALORDER[x]], TOTALORDER[inverse[x]]] == True

In[45]:= or[not[TOTALORDER[x_]], TOTALORDER[inverse[x_]]] := True
```

a few examples

In this section, some basic examples are given of total order relations as well as some examples of relations that are not.

```
In[46]:= TOTALORDER[0] // AssertTest

Out[46]= TOTALORDER[0] == True

In[47]:= TOTALORDER[0] := True

In[48]:= TOTALORDER[Id] // AssertTest

Out[48]= TOTALORDER[Id] == False

In[49]:= TOTALORDER[Id] := False

In[50]:= TOTALORDER[S] // AssertTest

Out[50]= TOTALORDER[S] == False

In[51]:= TOTALORDER[S] := False

In[52]:= TOTALORDER[id[x]] // AssertTest

Out[52]= TOTALORDER[id[x]] == or[equal[0, x], member[x, range[SINGLETON]]]

In[53]:= TOTALORDER[id[x_]] := or[equal[0, x], member[x, range[SINGLETON]]]

In[54]:= TOTALORDER[cart[x, x]] // AssertTest

Out[54]= TOTALORDER[cart[x, x]] == or[equal[0, x], member[x, range[SINGLETON]]]

In[55]:= TOTALORDER[cart[x_, x_]] := or[equal[0, x], member[x, range[SINGLETON]]]
```

Inclusion is a total order for the natural numbers. This provides an example of a total order that is an infinite set.

```
In[56]:= TOTALORDER[restrict[S, omega, omega]] // AssertTest
Out[56]= TOTALORDER[composite[id[omega], S, id[omega]]] == True

In[57]:= TOTALORDER[composite[id[omega], S, id[omega]]] := True
```

Inclusion is a total order for the ordinal numbers. This provides an example of a total order that is not a set.

```
In[58]:= TOTALORDER[restrict[S, OMEGA, OMEGA]] // AssertTest
Out[58]= TOTALORDER[composite[id[OMEGA], S, id[OMEGA]]] == True

In[59]:= TOTALORDER[composite[id[OMEGA], S, id[OMEGA]]] := True
```

The reflexive closure of the singleton of any pair is a total order. This shows that any ordered pair belongs to some total order relation.

```
In[60]:= TOTALORDER[union[cart[singleton[x], singleton[y]], id[pairset[x, y]]]] // AssertTest
Out[60]= TOTALORDER[union[cart[singleton[x], singleton[y]], id[pairset[x, y]]]] == True

In[61]:= TOTALORDER[union[cart[singleton[x_], singleton[y_]], id[pairset[x_, y_]]]] := True
```