wellorderings are transitive

Johan G. F. Belinfante
2004 March 14

```
In[1]:= << goedel55.09b; << tools.m;

        :Package Title: goedel55.09b        2004 March 9 at 12:15 noon

        It is now:  2004 Mar 15 at 12:45

        Loading Simplification Rules

        TOOLS.M                      Revised 2004 March 14

        weightlimit = 40
```

summary

An **Otter** proof that well orderings are transitive was obtained 2000 December 24. In this notebook, the fact that well−orderings are transitive is rederived in a technically different fashion, but still based on the key idea that a set of three elements each of which is fixed by the well−ordering has a least element. Three free variables are introduced to represent these elements, replacing Skolem functions that appear in the **Otter** proof. These introduced variables are eventually eliminated by using Gõdel's algorithm, which is formulated in the **GOEDEL** program as a set of rewrite rules for **class**. If no special measures are adopted, the **GOEDEL** program has no way of knowing whether these variables refer to sets or to proper classes, and rather complex expressions would be encountered to cover all cases. To circumvent this problem, an important new technique is introduced in this notebook, the concept of a **setpart** wrapper, which permits taking advantage of conditional rewrite rules that are valid for sets only. Nothing much in the way of new rewrite rules are needed for this new concept aside from its definition and the key property that **setpart[u]** is always a set. The **GOEDEL** program apparently finds it evident from the definition that every set can be wrapped with a **setpart** wrapper, and is able to remove these wrappers automatically with no special effort.

normalization results

The removal of set variables requires the use of **class**, which makes it imperative that normalization rules are in place to prevent the hypotheses from being rewritten. Two rules are needed, the first of which is needed to preserve the hypothesis that a relation be reflexive. A slightly more general rule is derived:

```
In[2]:= not[subclass[x, cart[y, z]]] // AssertTest // Reverse

Out[2]= or[not[subclass[x, cart[V, z]]], not[subclass[x, cart[y, V]]]] ==
         not[subclass[x, cart[y, z]]]

In[3]:= or[not[subclass[x_, cart[V, z_]]], not[subclass[x_, cart[y_, V]]]] :=
         not[subclass[x, cart[y, z]]]
```

The above rule affects the next one because the hypothesis that **x** be reflexive is a part of the definition of well−ordering.

```
In[4]:= not[WELLORDER[x]] // AssertTest // Reverse

Out[4]= or[not[subclass[x, cart[fix[x], fix[x]]]],
          not[subclass[P[fix[x]], union[domain[funpart[LEAST[x]]], singleton[0]]]]] == not[
          WELLORDER[x]]

In[5]:= or[not[subclass[x_, cart[fix[x_], fix[x_]]]],
          not[subclass[P[fix[x_]], union[domain[funpart[LEAST[x_]]], singleton[0]]]]] := not[
          WELLORDER[x]]
```

existence of least elements

There is a simple formula for **domain[LEAST[x]]** when **x** is a wellordering:

```
In[6]:= implies[WELLORDER[x], equal[domain[LEAST[x]], dif[P[fix[x]], singleton[0]]]]

Out[6]= True
```

In this section, a version of this formula is introduced with an extra variable, which will be needed shortly. To derive this formula, the following temporary lemma is needed.

```
In[7]:= SubstTest[implies, and[member[y, u], equal[u, v]], member[y, v],
            {u -> dif[P[fix[x]], singleton[0]], v -> domain[LEAST[x]]}]

Out[7]= or[equal[0, y], not[equal[0, intersection[y, lb[x, y]]]],
          not[equal[domain[LEAST[x]], intersection[complement[singleton[0]], P[fix[x]]]]],
          not[member[y, V]], not[subclass[y, fix[x]]]] == True

In[8]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

The hypothesis that **x** is a well−ordering is used to eliminate the literal involving **domain[LEAST[x]]** in the above statement.

```
In[9]:= Map[not, SubstTest[and, implies[p1, p2],
          implies[and[p2, p3], p4], not[implies[and[p1, p3], p4]], {p1 -> WELLORDER[x],
           p2 -> equal[domain[LEAST[x]], intersection[complement[singleton[0]], P[fix[x]]]],
                p3 -> and[not[equal[0, y]], member[y, V], subclass[y, fix[x]]],
           p4 -> not[equal[0, intersection[y, lb[x, y]]]]}]]

Out[9]= or[equal[0, y], not[equal[0, intersection[y, lb[x, y]]]],
          not[member[y, V]], not[subclass[y, fix[x]]], not[WELLORDER[x]]] == True

In[10]:= or[equal[0, y_], not[equal[0, intersection[y_, lb[x_, y_]]]],
           not[member[y_, V]], not[subclass[y_, fix[x_]]], not[WELLORDER[x_]]] := True
```

The following special formula for the least element of a singleton will be needed.

```
In[11]:= least[x, singleton[y]] // Normality

Out[11]= intersection[lb[x, singleton[y]], singleton[y]] == intersection[fix[x], singleton[y]]

In[12]:= intersection[lb[x_, singleton[y_]], singleton[y_]] :=
          intersection[fix[x], singleton[y]]
```

definition of the setpart wrapper

The concept of **setpart** is useful to wrap sets to permit automatic simplifications that apply only to sets. The following membership rule defines this wrapper:

```
In[13]:= member[x_, setpart[y_]] := and[member[x, y], member[y, V]]
```

The following normalization formula could be used to define **setpart** should one want to use this concept for automated reasoning using **Otter**. This normalization rule is not needed for the application in this notebook, and will not be added as a rewrite rule because doing so would cause **setpart** to be introduced in situations where it is not wanted, and one would then need to add many new rules to the **GOEDEL** program to eliminate it.

```
In[14]:= setpart[x] // Normality // Reverse

Out[14]= intersection[x, image[V, singleton[x]]] == setpart[x]
```

The omly important property of **setpart** is that it always be recognized to be a set when it is encountered by conditional rewrite rules.

```
In[15]:= Map[member[#, V] &, %] // Reverse

Out[15]= member[setpart[x], V] == True

In[16]:= member[setpart[x_], V] := True
```

key ideas

The key idea is that if **x** is a well−ordering, then any set of three elements of **fix[x]** must have a least member. It is convenient to introduce these three elements one at a time to prevent rules for **pairset** from entering into the formulas that are produced. Each of the variables will be wrapped with **setpart** to make it known that it is a set. The first element to be introduced is called **setpart[u]** .

```
In[17]:= SubstTest[implies, and[WELLORDER[x], member[w, dif[P[fix[x]], singleton[0]]]],
          not[empty[least[x, w]]]], w -> union[singleton[setpart[u]], z]]

Out[17]= or[not[equal[0, intersection[z, lb[x, z], lb[x, singleton[setpart[u]]]]]],
           not[member[z, V]], not[member[setpart[u], fix[x]]], not[subclass[z, fix[x]]],
           not[WELLORDER[x]], subclass[z, image[x, singleton[setpart[u]]]]] == True
```

The second element to be introduced is called **setpart[v]**.

```
In[18]:= % /. z -> union[y, singleton[setpart[v]]]

Out[18]= or[and[member[pair[setpart[u], setpart[v]], x],
            subclass[y, image[x, singleton[setpart[u]]]]],
           and[member[pair[setpart[v], setpart[u]], x],
            subclass[y, image[x, singleton[setpart[v]]]]],
           not[equal[0, intersection[y, lb[x, y], lb[x, singleton[setpart[u]]],
              lb[x, singleton[setpart[v]]]]]], not[member[y, V]],
           not[member[setpart[u], fix[x]]], not[member[setpart[v], fix[x]]],
           not[subclass[y, fix[x]]], not[WELLORDER[x]]] == True
```

The final variable is called **setpart[w]**.

```
In[19]:= % /. y -> singleton[setpart[w]]

Out[19]= or[and[member[pair[setpart[u], setpart[v]], x],
            member[pair[setpart[u], setpart[w]], x]],
          and[member[pair[setpart[v], setpart[u]], x],
            member[pair[setpart[v], setpart[w]], x]], and[
            member[pair[setpart[w], setpart[u]], x], member[pair[setpart[w], setpart[v]], x]],
          not[member[setpart[u], fix[x]]], not[member[setpart[v], fix[x]]],
          not[member[setpart[w], fix[x]]], not[WELLORDER[x]]] == True

In[20]:= (% /. {u -> u_, v -> v_, w -> w_, x -> x}) /. Equal -> SetDelayed
```

The result obtained above can be restated in the following fashion, which suggests a procedure for eliminating the variables **u**, **v** and **w**.

```
In[21]:= implies[WELLORDER[x], member[pair[pair[setpart[u], setpart[v]], setpart[w]],
          union[complement[cart[cart[fix[x], fix[x]], fix[x]]],
            composite[x, FIRST, id[x]], composite[x, SECOND, id[inverse[x]]],
            intersection[composite[inverse[x], FIRST], composite[inverse[x], SECOND]]]]]

Out[21]= True
```

eliminating the set variables

To eliminate the variables, it helps to turn off the **simplify** flag.

```
In[22]:= simplify = False;
```

The elimination of variables yields the following complicated mess, but note that **setpart** has been eliminated without having to introduce any special rewrite rules.

```
In[23]:= Map[equal[0, composite[complement[#], id[cart[V, V]]]] &,
         SubstTest[class, pair[pair[u, v], w],
           implies[WELLORDER[x], member[pair[pair[setpart[u], setpart[v]], setpart[w]], z]],
            z -> union[complement[cart[cart[fix[x], fix[x]], fix[x]]],
             composite[x, FIRST, id[x]], composite[x, SECOND, id[inverse[x]]], intersection[
              composite[inverse[x], FIRST], composite[inverse[x], SECOND]]]]] // Reverse]

Out[23]= or[and[subclass[
             cart[fix[x], intersection[fix[x], fix[composite[complement[x], id[fix[x]], x]]]],
             union[x, inverse[x]]], subclass[cart[fix[x], intersection[fix[x],
               fix[composite[complement[x], id[fix[x]], complement[x]]]]], x],
           subclass[composite[id[fix[x]], intersection[x, complement[inverse[x]]],
             id[fix[x]], complement[inverse[x]], id[fix[x]]], x], subclass[composite[
             id[fix[x]], intersection[complement[x], complement[inverse[x]]], id[fix[x]],
             intersection[x, inverse[x]], id[fix[x]]], x]], not[WELLORDER[x]]] == True
```

The conclusion in the above statement contains three assertions, which is more than one actually needs. It suffices to cut the conclusion down to one of these three:

```
In[24]:= Map[or[#, subclass[composite[id[fix[x]], intersection[x, complement[inverse[x]]],
              id[fix[x]], complement[inverse[x]], id[fix[x]]], x]] &, %]

Out[24]= or[not[WELLORDER[x]],
          subclass[composite[id[fix[x]], intersection[x, complement[inverse[x]]],
            id[fix[x]], complement[inverse[x]], id[fix[x]]], x]] == True

In[25]:= (% /. x -> x_) /. Equal -> SetDelayed
```

In the next section, special rules are derived to eliminate the expression **complement[inverse[x]]** that appears in this statement, effectively replacing that with the simpler expression **intersection[Di,x]**.

a theorem about inverse complements of well orderings

Lemma. (The fact that a well ordering **x** is reflexive is combined with the property that any two elements of **fix[x]** can be compared.)

```
In[26]:= Map[implies[WELLORDER[x], #] &, SubstTest[subclass, x, intersection[u, v],
            {u -> cart[fix[x], fix[x]], v -> union[complement[inverse[x]], Id]}]] // MapNotNot

Out[26]= or[not[WELLORDER[x]], subclass[x, union[
             composite[id[fix[x]], complement[inverse[x]], id[fix[x]]], id[fix[x]]]]] == True

In[27]:= (% /. x -> x_) /. Equal -> SetDelayed
```

Lemma. (Equality implies inclusion.)

```
In[28]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
               {p1 -> WELLORDER[x], p2 -> equal[cart[fix[x], fix[x]], union[x, inverse[x]]],
            p3 -> subclass[cart[fix[x], fix[x]], union[x, inverse[x]]]}]]

Out[28]= or[not[WELLORDER[x]], subclass[cart[fix[x], fix[x]], union[x, inverse[x]]]] == True

In[29]:= (% /. x -> x_) /. Equal -> SetDelayed
```

The above two inclusions can be combined into an equation expressing a well order **x** in terms of **complement[inverse[x]]**:

```
In[30]:= SubstTest[and, implies[p, subclass[x, y]], implies[p, subclass[y, x]],
             {p -> WELLORDER[x], y -> union[composite[id[fix[x]],
               complement[inverse[x]], id[fix[x]]], id[fix[x]]]}] // Reverse

Out[30]= or[equal[x, union[composite[id[fix[x]], complement[inverse[x]], id[fix[x]]],
             id[fix[x]]]], not[WELLORDER[x]]] == True

In[31]:= or[equal[x_, union[composite[id[fix[x_]], complement[inverse[x_]], id[fix[x_]]],
             id[fix[x_]]]], not[WELLORDER[x_]]] := True
```

The **simplify** flag needs to be turned back on now.

```
In[32]:= simplify = True;
```

Lemma. (Eliminating an unneeded intersection with **Di**.)

```
In[33]:= composite[id[fix[x]],
           intersection[Di, complement[inverse[x]]], id[fix[x]]] // VSNormality

Out[33]= composite[id[fix[x]], intersection[Di, complement[inverse[x]]], id[fix[x]]] ==
          composite[id[fix[x]], complement[inverse[x]], id[fix[x]]]

In[34]:= composite[id[fix[x_]], intersection[Di, complement[inverse[x_]]], id[fix[x_]]] :=
          composite[id[fix[x]], complement[inverse[x]], id[fix[x]]]
```

Lemma. (The equation derived above for a well order **x** is solved for **complement[inverse[x]]** by intersecting with **Di** .)

```
In[35]:= SubstTest[implies, equal[x, y], equal[image[z, x], image[z, y]],
           {y -> union[composite[id[fix[x]], complement[inverse[x]], id[fix[x]]],
           id[fix[x]]], z -> id[Di]}]

Out[35]= or[equal[composite[id[fix[x]], complement[inverse[x]], id[fix[x]]],
            intersection[Di, x]], not[equal[x, union[
              composite[id[fix[x]], complement[inverse[x]], id[fix[x]]], id[fix[x]]]]]] == True

In[36]:= (% /. x -> x_) /. Equal -> SetDelayed
```

Main result: (The expression **complement[inverse[x]]** can be replaced with **intersection[Di,x]**.)

```
In[37]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
               {p1 -> WELLORDER[x],
          p2 -> equal[x,
             union[composite[id[fix[x]], complement[inverse[x]], id[fix[x]]], id[fix[x]]]],
                    p3 -> equal[composite[id[fix[x]], complement[inverse[x]], id[fix[x]]],
             intersection[Di, x]]}]]

Out[37]= or[equal[composite[id[fix[x]], complement[inverse[x]], id[fix[x]]],
            intersection[Di, x]], not[WELLORDER[x]]] == True

In[38]:= or[equal[composite[id[fix[x_]], complement[inverse[x_]], id[fix[x_]]],
            intersection[Di, x_]], not[WELLORDER[x_]]] := True
```

A variant is also needed.

```
In[39]:= SubstTest[implies, equal[u, v], equal[image[w, u], image[w, v]],
           {u -> composite[id[fix[x]], complement[inverse[x]], id[fix[x]]],
          v -> intersection[Di, x], w -> id[x]}]

Out[39]= or[equal[composite[id[fix[x]], intersection[x, complement[inverse[x]]], id[fix[x]]],
            intersection[Di, x]],
          not[equal[composite[id[fix[x]], complement[inverse[x]], id[fix[x]]],
            intersection[Di, x]]]] == True

In[40]:= (% /. x -> x_) /. Equal -> SetDelayed

In[41]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
               {p1 -> WELLORDER[x], p2 -> equal[
             composite[id[fix[x]], complement[inverse[x]], id[fix[x]]], intersection[Di, x]],
                    p3 -> equal[composite[id[fix[x]], intersection[x, complement[inverse[x]]],
              id[fix[x]]], intersection[Di, x]]}]]

Out[41]= or[equal[composite[id[fix[x]], intersection[x, complement[inverse[x]]], id[fix[x]]],
            intersection[Di, x]], not[WELLORDER[x]]] == True

In[42]:= (% /. x -> x_) /. Equal -> SetDelayed
```

Equality substitution for composites is used:

```
In[43]:= SubstTest[implies, and[equal[u, v], equal[w, z]],
          equal[composite[u, w], composite[v, z]],
            {u -> composite[id[fix[x]], intersection[x, complement[inverse[x]]], id[fix[x]]],
            w -> composite[id[fix[x]], complement[inverse[x]], id[fix[x]]],
            v -> intersection[Di, x], z -> intersection[Di, x]}]

Out[43]= or[equal[composite[intersection[Di, x], intersection[Di, x]],
            composite[id[fix[x]], intersection[x, complement[inverse[x]]],
             id[fix[x]], complement[inverse[x]], id[fix[x]]]], not[equal[
             composite[id[fix[x]], complement[inverse[x]], id[fix[x]]], intersection[Di, x]]],
          not[equal[composite[id[fix[x]], intersection[x, complement[inverse[x]]],
              id[fix[x]]], intersection[Di, x]]]] == True
```

```
In[44]:= (% /. x -> x_) /. Equal -> SetDelayed
```

the final steps

A special lemma is needed to take advantage of the following rewrite rule:

```
In[45]:= subclass[composite[intersection[Di, x], intersection[Di, x]], x]

Out[45]= subclass[composite[x, x], x]
```

Lemma.

```
In[46]:= SubstTest[implies, and[equal[u, v], subclass[v, w]], subclass[u, w],
            {u -> composite[intersection[Di, x], intersection[Di, x]],
          v -> composite[id[fix[x]], intersection[x, complement[inverse[x]]],
            id[fix[x]], complement[inverse[x]], id[fix[x]]], w -> x}]

Out[46]= or[not[equal[composite[intersection[Di, x], intersection[Di, x]],
            composite[id[fix[x]], intersection[x, complement[inverse[x]]],
             id[fix[x]], complement[inverse[x]], id[fix[x]]]]],
          not[subclass[composite[id[fix[x]], intersection[x, complement[inverse[x]]],
             id[fix[x]], complement[inverse[x]], id[fix[x]]], x]],
          subclass[composite[x, x], x]] == True

In[47]:= (% /. x -> x_) /. Equal -> SetDelayed
```

Main Theorem: well orderings are transitive.

```
In[48]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
          implies[p1, p4], implies[and[p3, p4], p5], implies[and[p2, p5], p6],
                not[implies[p1, p6]],
            {p1 -> WELLORDER[x],
           p2 -> subclass[composite[id[fix[x]], intersection[x, complement[inverse[x]]],
              id[fix[x]], complement[inverse[x]], id[fix[x]]], x],
                p3 -> equal[composite[id[fix[x]], complement[inverse[x]], id[fix[x]]],
             intersection[Di, x]],
                p4 -> equal[composite[id[fix[x]], intersection[x, complement[inverse[x]]],
              id[fix[x]]], intersection[Di, x]],
                p5 -> equal[composite[id[fix[x]], intersection[x, complement[inverse[x]]],
              id[fix[x]], complement[inverse[x]], id[fix[x]]], composite[
              intersection[Di, x], intersection[Di, x]]], p6 -> subclass[composite[x, x], x]}]]

Out[48]= or[not[WELLORDER[x]], subclass[composite[x, x], x]] == True

In[49]:= or[not[WELLORDER[x_]], subclass[composite[x_, x_], x_]] := True
```

Corollary:

```
In[50]:= Map[not, SubstTest[and, implies[p1, p2],
          implies[p1, p3], implies[and[p2, p3], p4], not[implies[p1, p4]],
                {p1 -> WELLORDER[x], p2 -> subclass[x, cart[V, V]],
           p3 -> subclass[composite[x, x], x], p4 -> TRANSITIVE[x]}]]

Out[50]= or[not[WELLORDER[x]], TRANSITIVE[x]] == True

In[51]:= or[not[WELLORDER[x_]], TRANSITIVE[x_]] := True
```

A variable−free version of this can be derived for the special case of small well−orderings:

```
In[52]:= Map[equal[V, #] &, SubstTest[class, x, not[member[x, y]], y -> dif[WO, TRV]]] // Reverse

Out[52]= subclass[WO, TRV] == True

In[53]:= subclass[WO, TRV] := True
```

Not all well orderings are sets. For example, the restriction of the subclass relation **S** to the class **OMEGA** of all ordinals is a well ordering which is a proper class.

```
In[54]:= WELLORDER[restrict[S, OMEGA, OMEGA]]

Out[54]= True
```

some further corollaries

In this final section, the results obtained above are combined with other previously derived facts about well orderings to derive various useful corollaries.

```
In[55]:= implies[WELLORDER[x], PARTIALORDER[x]] // NotNotTest

Out[55]= or[and[subclass[x, cart[fix[x], fix[x]]],
            subclass[intersection[x, inverse[x]], Id], TRANSITIVE[x]], not[WELLORDER[x]]] == True

In[56]:= (% /. x -> x_) /. Equal -> SetDelayed

In[57]:= Map[equal[V, #] &, SubstTest[class, x, not[member[x, y]], y -> dif[WO, PO]]] // Reverse

Out[57]= subclass[WO, PO] == True

In[58]:= subclass[WO, PO] := True
```

If **x** is a well ordering, then **intersection[Di,x]** is acyclic.

```
In[59]:= SubstTest[implies, and[subclass[x, y], subclass[y, z]], subclass[x, z],
             {x -> WO, y -> PO, z -> image[inverse[IMAGE[id[Di]]], ACYCLIC]}]

Out[59]= subclass[image[IMAGE[id[Di]], WO], ACYCLIC] == True

In[60]:= subclass[image[IMAGE[id[Di]], WO], ACYCLIC] := True
```

Well orderings are total orderings.

```
In[61]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
           not[implies[p1, p4]], {p1 -> WELLORDER[x], p2 -> PARTIALORDER[x],
            p3 -> equal[cart[fix[x], fix[x]], union[x, inverse[x]]], p4 -> TOTALORDER[x]}]]

Out[61]= or[not[WELLORDER[x]], TOTALORDER[x]] == True

In[62]:= or[not[WELLORDER[x_]], TOTALORDER[x_]] := True

In[63]:= Map[equal[V, #] &, SubstTest[class, x, not[member[x, y]], y -> dif[WO, TO]]] // Reverse

Out[63]= subclass[WO, TO] == True

In[64]:= subclass[WO, TO] := True
```