wellfounded induction

Johan G. F. Belinfante
2003 October 6

```
In[1]:= << goedel52.s99; << tools.m

        :Package Title: goedel52.s99      2003 September 29 at 8:15 p.m.

        It is now:  2003 Oct 6 at 9:36

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 September 29

        weightlimit = 40
```

summary

This notebook establishes the principle of well−founded induction: it is shown that if **x** is a well−founded relation whose inverse is thin, then the only class which is invariant under **x** is the empty class. The reason for calling this a form of induction has to do with a certain way of applying this theorem; the idea is that if one wants to show that some statement is true for all sets, one could begin by letting **y** be the class of all counterexamples, and then attempt to show that if **z** is any counterexample, there is an **x**−prior counterexample, so that the class **y** of counterexamples would be subvariant under **x**, and therefore empty. The techniques used in this notebook were inspired by a proof given in a book by Azriel Levy, called *Basic Set Theory*, modified to conform with the terminology and techniques available in the **GOEDEL** program.

As an application, it is shown that if **x** is a well−founded relation whose inverse is thin, then so is its transitive closure **trv[x]**. This solves an exercise posed in Levy's book.

normalizing the predicate WELLFOUNDED

Normalizing the predicate **WELLFOUNDED** results in simpler proofs. The idea is to prevent applications of **AssertTest** from disturbing this predicate.

```
In[2]:= WELLFOUNDED[composite[Id, x]] // AssertTest // Reverse

Out[2]= subclass[subvar[x], singleton[0]] == WELLFOUNDED[composite[Id, x]]

In[3]:= subclass[subvar[x_], singleton[0]] := WELLFOUNDED[composite[Id, x]]

In[4]:= WELLFOUNDED[x] // AssertTest // Reverse

Out[4]= and[subclass[x, cart[V, V]], WELLFOUNDED[composite[Id, x]]] == WELLFOUNDED[x]

In[5]:= and[subclass[x_, cart[V, V]], WELLFOUNDED[composite[Id, x_]]] := WELLFOUNDED[x]
```

a slightly wierd beginning

One deviation of the present derivation from that presented in Levy's book is that a conscious effort was made to avoid indirect proof and to avoid introducing unneeded variables. To reduce the number of variables, the following slightly wierd trick was used. The statement that **z** is invariant under **inverse[x]** is logically equivalent to a subcommutativity statement. In fact the **GOEDEL** program automatically rewrites the latter to the former:

```
In[6]:= equiv[invariant[inverse[x], z], subcommute[id[z], x]]

Out[6]= True
```

This equivalence was used to establish the following lemma used to avoid some variables produced via Skolemizing existential quantifiers in Levy's proof.

```
In[7]:= SubstTest[implies, subclass[u, v], subclass[image[u, y], image[v, y]],
            {u -> composite[id[z], x], v -> composite[x, id[z]]}]

Out[7]= or[not[subclass[image[inverse[x], z], z]],
          subclass[intersection[z, image[x, y]], image[x, intersection[y, z]]]] == True

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

Restatement:

```
In[9]:= implies[invariant[inverse[x], z],
         subclass[intersection[z, image[x, y]], image[x, intersection[y, z]]]]

Out[9]= True
```

In particular, this result can be applied the class **z = range[iterate[inverse[x],w]]**, which satisfies the following invariance property:

```
In[10]:= invariant[inverse[x], range[iterate[inverse[x], w]]]

Out[10]= True
```

This application yields the following result:

```
In[11]:= SubstTest[implies, invariant[inverse[x], w],
          subclass[intersection[w, image[x, y]], image[x, intersection[w, y]]],
             w -> range[iterate[inverse[x], z]]]

Out[11]= subclass[intersection[image[x, y], range[iterate[inverse[x], z]]],
            image[x, intersection[y, range[iterate[inverse[x], z]]]]] == True

In[12]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed
```

Suppose now that **y** is subvariant under **x**. Then the above statement implies that **intersection[y,range[iterate[inverse[x],z]]]** is also subvariant under **x**.

```
In[13]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u -> intersection[y, range[iterate[inverse[x], z]]],
             v -> intersection[image[x, y], range[iterate[inverse[x], z]]],
             w -> image[x, intersection[y, range[iterate[inverse[x], z]]]]}]

Out[13]= or[not[subclass[intersection[y, range[iterate[inverse[x], z]]], image[x, y]]],
           subclass[intersection[y, range[iterate[inverse[x], z]]],
            image[x, intersection[y, range[iterate[inverse[x], z]]]]]] == True

In[14]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed

In[15]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p2, p3], not[implies[p1, p3]], {p1 -> subvariant[x, y],
             p2 -> subclass[intersection[y, range[iterate[inverse[x], z]]], image[x, y]],
          p3 -> subvariant[x, intersection[y, range[iterate[inverse[x], z]]]]}]]

Out[15]= or[not[subclass[y, image[x, y]]],
           subclass[intersection[y, range[iterate[inverse[x], z]]],
            image[x, intersection[y, range[iterate[inverse[x], z]]]]]] == True

In[16]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed
```

Restatement:

```
In[17]:= implies[subvariant[x, y],
           subvariant[x, intersection[y, range[iterate[inverse[x], z]]]]]

Out[17]= True
```

The intension is to use this result to prove that if **x** is wellfounded, and **inverse[x]** is thin, then the only class subvariant under **x** is the empty set. The definition of a wellfounded relation already implies that this is the case for subvariant sets. The thin−ness hypothesis will be used to extend this to subvariant classes in general. The thin−ness hypothesis is used via the following lemma:

```
In[18]:= SubstTest[implies, and[thin[u], member[v, V]],
           member[iterate[u, v], V], {u -> inverse[x], v -> singleton[z]}]

Out[18]= or[member[iterate[inverse[x], singleton[z]], V],
           not[equal[V, domain[VERTSECT[inverse[x]]]]]] == True

In[19]:= (% /. {x -> x_, z -> z_}) /. Equal -> SetDelayed
```

Restatement:

```
In[20]:= implies[thin[inverse[x]], member[iterate[inverse[x], singleton[z]], V]]

Out[20]= True
```

To apply this to the current situation, one needs to add an intersection with **y** and **range**, which is done by using the axiom of replacement:

```
In[21]:= SubstTest[implies, and[FUNCTION[w], member[x, V]],
           member[image[w, x], V], w -> composite[id[y], SECOND]]

Out[21]= or[member[intersection[y, range[x]], V], not[member[x, V]]] == True

In[22]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

These facts need to be assembled:

```
In[23]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 -> thin[inverse[x]], p2 -> member[iterate[inverse[x], singleton[z]], V],
            p3 ->
          member[intersection[y, range[iterate[inverse[x], singleton[z]]]], V]}]]

Out[23]= or[member[intersection[y, range[iterate[inverse[x], singleton[z]]]], V],
          not[equal[V, domain[VERTSECT[inverse[x]]]]]] == True

In[24]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed

In[25]:= implies[and[thin[inverse[x]], subvariant[x, y]],
           member[intersection[y, range[iterate[inverse[x], singleton[z]]]], subvar[x]]] //
         NotNotTest

Out[25]= or[and[member[intersection[y, range[iterate[inverse[x], singleton[z]]]], V],
           subclass[intersection[y, range[iterate[inverse[x], singleton[z]]]],
            image[x, intersection[y, range[iterate[inverse[x], singleton[z]]]]]]],
          not[equal[V, domain[VERTSECT[inverse[x]]]]], not[subclass[y, image[x, y]]]] == True

In[26]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed
```

well−foundedness

Recall:

```
In[27]:= implies[WELLFOUNDED[x], equal[subvar[x], singleton[0]]]

Out[27]= True
```

Here we need this with an additional variable:

```
In[28]:= SubstTest[implies, and[member[y, u], equal[u, v]],
          member[y, v], {u -> subvar[x], v -> singleton[0]}]

Out[28]= or[equal[0, y], not[equal[singleton[0], subvar[x]]],
          not[member[y, V]], not[subclass[y, image[x, y]]]] == True

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

Some assembly is required:

```
In[30]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
            {p1 -> WELLFOUNDED[x], p2 -> equal[subvar[x], singleton[0]],
           p3 -> implies[and[member[y, V], subvariant[x, y]], equal[0, y]]}]]

Out[30]= or[equal[0, y], not[member[y, V]],
          not[subclass[y, image[x, y]]], not[WELLFOUNDED[x]]] == True

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

The whole business must now be combined:

```
In[32]:= Map[not, SubstTest[and, implies[and[p1, p4], p5],
           implies[and[p2, p3], p4], not[implies[and[p1, p2, p3], p5]],
             {p1 -> WELLFOUNDED[x], p2 -> thin[inverse[x]], p3 -> subvariant[x, y],
              p4 -> member[intersection[y, range[iterate[inverse[x], singleton[z]]]], subvar[x]],
                   p5 -> equal[0, intersection[y, range[iterate[inverse[x], singleton[z]]]]]}]]

Out[32]= or[equal[0, intersection[y, range[iterate[inverse[x], singleton[z]]]]],
           not[equal[V, domain[VERTSECT[inverse[x]]]]],
           not[subclass[y, image[x, y]]], not[WELLFOUNDED[x]]] == True

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

Restatement:

```
In[34]:= implies[and[WELLFOUNDED[x], thin[inverse[x]], subvariant[x, y]],
             equal[0, intersection[y, range[iterate[inverse[x], singleton[z]]]]]]

Out[34]= True
```

To complete the argument one needs to note that if **y** is not zero once can pick **z** to be an element of **y**, and this will belong to the set **intersection[y,...]**. Indeed:

```
In[35]:= Map[or[not[member[z, y]], #] &,
           SubstTest[implies, and[member[z, s], subclass[s, t]], member[z, t],
              {s -> singleton[z], t -> range[iterate[inverse[x], singleton[z]]]}]]

Out[35]= or[member[z, range[iterate[inverse[x], singleton[z]]]], not[member[z, y]]] == True

In[36]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed

In[37]:= implies[member[z, y],
           member[z, intersection[y, range[iterate[inverse[x], singleton[z]]]]]]

Out[37]= True

In[38]:= SubstTest[implies, and[member[z, u], equal[u, v]], member[z, v],
             {u -> intersection[y, range[iterate[inverse[x], singleton[z]]]], v -> 0}]

Out[38]= or[not[equal[0, intersection[y, range[iterate[inverse[x], singleton[z]]]]]],
           not[member[z, y]], not[member[z, range[iterate[inverse[x], singleton[z]]]]]] == True

In[39]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed
```

the final fort

At this point one is almost done. The explicit occurrence of the iteration construction is eliminated:

```
In[40]:= Map[not, SubstTest[and, implies[and[p1, p2, p3], p4], implies[p5, p6],
           implies[and[p5, p6], not[p4]], not[implies[and[p1, p2, p3], not[p5]]],
             {p1 -> WELLFOUNDED[x], p2 -> thin[inverse[x]], p3 -> subvariant[x, y],
              p4 -> equal[0, intersection[y, range[iterate[inverse[x], singleton[z]]]]],
                   p5 -> member[z, y], p6 -> member[z, range[iterate[inverse[x], singleton[z]]]]}]]

Out[40]= or[not[equal[V, domain[VERTSECT[inverse[x]]]]], not[member[z, y]],
           not[subclass[y, image[x, y]]], not[WELLFOUNDED[x]]] == True

In[41]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed
```

The final problem is to eliminate the variable **z**. Although this is conceptually simple, the trick was to exploit the following fact to sequester the predicate **WELLFOUNDED** from the action of **class**:

```
In[42]:= subclass[P[x], WF]

Out[42]= WELLFOUNDED[x]
```

This completes the derivation of the theorem about well−founded induction:

```
In[43]:= Map[equal[V, #] &, SubstTest[class, z,
           or[not[member[z, y]], not[equal[V, domain[VERTSECT[inverse[x]]]]],
            not[subclass[y, image[x, y]]], not[subclass[P[x], w]]], w -> WF]] // Reverse

Out[43]= or[equal[0, y], not[equal[V, domain[VERTSECT[inverse[x]]]]],
           not[subclass[y, image[x, y]]], not[WELLFOUNDED[x]]] == True

In[44]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

Restatement:

```
In[45]:= implies[and[WELLFOUNDED[x], thin[inverse[x]], subvariant[x, y]], equal[0, y]]

Out[45]= True
```

applications

These are some corollaries of the theorem about well−founded induction.

```
In[46]:= SubstTest[implies, and[WELLFOUNDED[w], thin[inverse[w]]],
          implies[subvariant[w, x], equal[0, x]], w → composite[id[FINITE], PS]]

Out[46]= or[equal[0, x], not[subclass[x, FINITE]], not[subclass[x, image[PS, x]]]] == True

In[47]:= or[equal[0, x_], not[subclass[x_, FINITE]], not[subclass[x_, image[PS, x_]]]] := True

In[48]:= SubstTest[implies, and[WELLFOUNDED[w], thin[inverse[w]], subvariant[w, y]],
          equal[0, y], {w → composite[id[REGULAR], E], y → complement[x]}]

Out[48]= or[equal[V, x], not[equal[V, union[REGULAR, x]]], not[subclass[P[x], x]]] == True

In[49]:= SubstTest[implies, and[WELLFOUNDED[w], thin[inverse[w]], subvariant[w, y]],
          equal[0, y], {w → composite[id[REGULAR], E], y → dif[REGULAR, x]}]

Out[49]= or[not[subclass[P[intersection[REGULAR, x]], x]], subclass[REGULAR, x]] == True
```

In particular if the axiom of regularity is assumed, then **V** is the only class which contains its own power class:

```
In[50]:= SubstTest[implies, and[WELLFOUNDED[w], thin[inverse[w]], subvariant[w, y]],
          equal[0, y], {w → composite[id[REGULAR], E], y -> complement[P[x]]}]

Out[50]= or[equal[V, x], not[equal[REGULAR, V]], not[subclass[P[x], x]]] == True
```

transitive closure theorem

The following two rewrite rules obscure the argument, and will be temporarily removed:

```
In[51]:= image[trv[x], y]

Out[51]= image[iterate[x, y], complement[singleton[0]]]

In[52]:= image[trv[x_], y_] =.

In[53]:= equal[0, image[x, y]]

Out[53]= equal[0, intersection[y, domain[x]]]

In[54]:= equal[0, image[x_, y_]] =.

In[55]:= ImageComp[x, trv[x], y]

Out[55]= image[trv[x], image[trv[x], y]] == image[x, image[trv[x], y]]

In[56]:= image[trv[x_], image[trv[x_], y_]] := image[x, image[trv[x], y]]

In[57]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
             {u -> y, v -> image[trv[x], y], w -> trv[x]}]

Out[57]= or[not[subclass[y, image[trv[x], y]]],
           subclass[image[trv[x], y], image[x, image[trv[x], y]]]] == True

In[58]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

Restatement:

```
In[59]:= implies[subvariant[trv[x], y], subvariant[x, image[trv[x], y]]]

Out[59]= True

In[60]:= Map[not, SubstTest[and, implies[and[p1, p2, p4], p5], implies[p3, p4],
           implies[and[p3, p5], p6], not[implies[and[p1, p2, p3], p6]],
           {p1 -> WELLFOUNDED[x], p2 -> thin[inverse[x]], p3 -> subvariant[trv[x], y],
             p4 -> subvariant[x, image[trv[x], y]],
            p5 -> equal[0, image[trv[x], y]], p6 -> equal[0, y]}]]

Out[60]= or[equal[0, y], not[equal[V, domain[VERTSECT[inverse[x]]]]],
           not[subclass[y, image[trv[x], y]]], not[WELLFOUNDED[x]]] == True

In[61]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

The trick for sequestering the predicate **WELLFOUNDED** is used once again:

```
In[62]:= Map[equal[V, #] &, SubstTest[class, y, or[equal[0, y],
             not[equal[V, domain[VERTSECT[inverse[x]]]]], not[subclass[y, image[t, y]]],
             not[subclass[P[x], w]]], {t -> trv[x], w -> WF}]] // Reverse

Out[62]= or[not[equal[V, domain[VERTSECT[inverse[x]]]]],
           not[WELLFOUNDED[x]], WELLFOUNDED[trv[x]]] == True
```

This rewrite rule says that if a well−founded relation has a thin inverse, then the transitive closure is well founded.

```
In[63]:= or[not[equal[V, domain[VERTSECT[inverse[x_]]]]],
            not[WELLFOUNDED[x_]], WELLFOUNDED[trv[x_]]] := True
```

Thw transitive closure of a thin relation is thin, so a similar statement holds for inverses of such relations:

```
In[64]:= SubstTest[implies, thin[y], thin[trv[y]], y → inverse[x]]

Out[64]= or[equal[V, domain[VERTSECT[inverse[trv[x]]]]],
            not[equal[V, domain[VERTSECT[inverse[x]]]]] == True

In[65]:= or[equal[V, domain[VERTSECT[inverse[trv[x_]]]]],
            not[equal[V, domain[VERTSECT[inverse[x_]]]]] := True
```

some examples of well−founded relations

Does the normalization of **WELLFOUNDED** cause problems for examples? No; here are some examples:

```
In[66]:= WELLFOUNDED[cart[x, y]] // AssertTest

Out[66]= WELLFOUNDED[cart[x, y]] == equal[0, intersection[x, y]]

In[67]:= WELLFOUNDED[cart[x_, y_]] := equal[0, intersection[x, y]]

In[68]:= equiv[and[not[member[x, V]], not[member[first[x], V]]], not[member[x, V]]]

Out[68]= True

In[69]:= Equal[and[not[member[x, V]], not[member[first[x], V]]], not[member[x, V]]]

Out[69]= and[not[member[x, V]], not[member[first[x], V]]] == not[member[x, V]]

In[70]:= and[not[member[x_, V]], not[member[first[x_], V]]] := not[member[x, V]]

In[71]:= WELLFOUNDED[singleton[x]] // AssertTest

Out[71]= WELLFOUNDED[singleton[x]] ==
          or[and[member[first[x], V], not[equal[first[x], second[x]]]], not[member[x, V]]]

In[72]:= WELLFOUNDED[singleton[x_]] :=
          or[and[member[first[x], V], not[equal[first[x], second[x]]]], not[member[x, V]]]
```

variable−free reformulations of the examples

The examples considered in the preceding section can be recast in a variable−free form.

```
In[73]:= image[inverse[CART], WF] // RelnNormality

Out[73]= image[inverse[CART], WF] == DISJOINT

In[74]:= image[inverse[CART], WF] := DISJOINT

In[75]:= ImageComp[CART, inverse[CART], WF]

Out[75]= intersection[WF, range[CART]] == image[CART, DISJOINT]
```

```
In[76]:= intersection[WF, range[CART]] := image[CART, DISJOINT]

In[77]:= image[inverse[SINGLETON], WF] // Normality

Out[77]= image[inverse[SINGLETON], WF] == Di

In[78]:= image[inverse[SINGLETON], WF] := Di

In[79]:= ImageComp[SINGLETON, inverse[SINGLETON], WF]

Out[79]= intersection[WF, range[SINGLETON]] == image[SINGLETON, Di]

In[80]:= intersection[WF, range[SINGLETON]] := image[SINGLETON, Di]
```

Some related results:

```
In[81]:= composite[inverse[SINGLETON], CART] // TriNormality

Out[81]= composite[inverse[SINGLETON], CART] == cross[inverse[SINGLETON], inverse[SINGLETON]]

In[82]:= composite[inverse[SINGLETON], CART] := cross[inverse[SINGLETON], inverse[SINGLETON]]

In[83]:= composite[inverse[CART], SINGLETON] // DoubleInverse

Out[83]= composite[inverse[CART], SINGLETON] == cross[SINGLETON, SINGLETON]

In[84]:= composite[inverse[CART], SINGLETON] := cross[SINGLETON, SINGLETON]

In[85]:= ImageComp[inverse[SINGLETON], CART, DISJOINT] // Reverse

Out[85]= image[inverse[SINGLETON], image[CART, DISJOINT]] == Di

In[86]:= image[inverse[SINGLETON], image[CART, DISJOINT]] := Di

In[87]:= ImageComp[SINGLETON, inverse[SINGLETON], image[CART, DISJOINT]]

Out[87]= intersection[image[CART, DISJOINT], range[SINGLETON]] == image[SINGLETON, Di]

In[88]:= intersection[image[CART, DISJOINT], range[SINGLETON]] := image[SINGLETON, Di]
```

unneeded lemmas

The following rewite rules were used to help avoid wierd expressions in the proof of the result about transitive closure before the sequestering trick was discovered. They are no longer needed, but may be of interest for other applications.

```
In[89]:= composite[complement[inverse[E]], IMAGE[SWAP], id[TRV], S] // RelnNormality

Out[89]= composite[complement[inverse[E]], IMAGE[SWAP], id[TRV], S] ==
          composite[complement[inverse[E]], INVERSE, HULL[TRV]]

In[90]:= composite[complement[inverse[E]], IMAGE[SWAP], id[TRV], S] :=
          composite[complement[inverse[E]], INVERSE, HULL[TRV]]
```

```
In[91]:= ImageComp[IMAGE[SWAP], id[P[cart[V, V]]], x] // Reverse
Out[91]= image[IMAGE[SWAP], intersection[x, P[cart[V, V]]]] == image[INVERSE, x]

In[92]:= image[IMAGE[SWAP], intersection[x_, P[cart[V, V]]]] := image[INVERSE, x]

In[93]:= SubstTest[U, image[IMAGE[SWAP], y], y -> intersection[P[cart[V, V]], x]]
Out[93]= U[image[INVERSE, x]] == inverse[U[intersection[x, P[cart[V, V]]]]]

In[94]:= U[image[INVERSE, x_]] := inverse[U[intersection[x, P[cart[V, V]]]]]

In[95]:= ImageComp[id[P[cart[V, V]]], HULL[TRV], x] // Reverse
Out[95]= intersection[image[HULL[TRV], x], P[cart[V, V]]] == image[HULL[TRV], x]

In[96]:= intersection[image[HULL[TRV], x_], P[cart[V, V]]] := image[HULL[TRV], x]
```