listing ordinals in order leaves no gaps

Johan G. F. Belinfante
2006 May 31

```
In[1]:= SetDirectory["l:"]; << goedel81.29a; << tools.m

        :Package Title: goedel81.29a         2006 May 29 at 7:20 p.m.

        It is now:  2006 May 31 at 19:36

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 May 8

        weightlimit = 40
```

summary

The function **ordlist[x]** attempts to iteratively list the ordinals in a class **x** in increasing order. If there are no ordinals in the class **x**, then **ordlist[x]** is empty. In general, the domain of **ordlist[x]** is the length of the list, which is a natural number when the class **x** holds only finitely many ordinals, and is otherwise the set **omega** of all natural numbers. If there are infinitely many ordinals in the class **x**, the listing process goes on forever, but may fail to list all the ordinals in **x**. The definition of the function **ordlist[x]** is such that if some ordinal **w** has been listed, and if there are greater ordinals that belong to **x**, then the list goes on by listing the next ordinal the least ordinal in **x** strictly greater than **w**. This fact is expressed by the rewrite rule

```
In[2]:= hull[intersection[OMEGA, x], succ[APPLY[ordlist[x], y]]]

Out[2]= APPLY[ordlist[x], succ[y]]
```

In this notebook ordinary induction is used to show that the listing process leaves no gaps in the sense that whenever an ordinal is placed on the list, all lesser ordinals in the class **x** have already been listed previously. In the first part of this notebook, an extra variable **y** is used to denote where a given ordinal occurs in the list. This is done to facilitate the use of mathematical induction. Later, this variable **y** is eliminated to obtain a formulation of the no-gap property in terms of the class **x** alone. At the end of the notebook a corollary is derived about the range of **ordlist[x]**. The meaning of this corollary is clarified by re-introducing an extra variable **w**, denoting a particular member of the range.

base case

If there are no ordinals in the class **x**, then **ordlist[x]** is empty, and there is nothing to prove. Otherwise, the first ordinal to be listed is the least one in **x**.

```
In[3]:= APPLY[ordlist[x], 0]
Out[3]= A[intersection[OMEGA, x]]
```

In this case there are no lesser ordinals in the class **x**.

```
In[4]:= SubstTest[implies, subclass[y, REGULAR], disjoint[y, A[y]], y → intersection[OMEGA, x]]
Out[4]= equal[0, intersection[OMEGA, x, A[intersection[OMEGA, x]]]] == True

In[5]:= intersection[OMEGA, x_, A[intersection[OMEGA, x_]]] := 0
```

sethood lemmas

Since **ordlist** is a function whose domain is a set, it follows from the axiom of replacement that any class equal to one of its images must also be a set.

```
In[6]:= equiv[and[equal[z, image[ordlist[x], y]], member[z, V]], equal[z, image[ordlist[x], y]]]
Out[6]= True

In[7]:= and[equal[z_, image[ordlist[x_], y_]], member[z_, V]] := equal[z, image[ordlist[x], y]]
```

Also, none of its images can be equal to the universal class **V**, because the universal class is not a set.

```
In[8]:= SubstTest[equal, V, setpart[z], z → image[ordlist[x], y]]
Out[8]= equal[V, image[ordlist[x], y]] == False

In[9]:= equal[V, image[ordlist[x_], y_]] := False
```

These lemmas are needed to derive the membership rule in the next section.

a membership rule for the class used in the induction

When the **GOEDEL** program is used to prove statements about natural numbers using induction, one needs first to construct the class of numbers for which the statement is true and a membership rule for this class must be available. The needed membership rule is derived in this section.

```
In[10]:= (member[pair[u, v], composite[inverse[funpart[w]], funpart[z]]] // AssertTest) /.
          {w → IMAGE[ordlist[x]], z → composite[IMAGE[id[intersection[OMEGA, x]]], ordlist[x]]}
Out[10]= member[pair[u, v], composite[inverse[IMAGE[ordlist[x]]],
           IMAGE[id[intersection[OMEGA, x]]], ordlist[x]]] ==
         and[equal[image[ordlist[x], v], intersection[OMEGA, x, APPLY[ordlist[x], u]]],
          member[u, domain[ordlist[x]]], member[v, V]]
```

```
In[11]:= member[pair[u_, v_], composite[inverse[IMAGE[ordlist[x_]]],
           IMAGE[id[intersection[OMEGA, x_]]], ordlist[x_]]] :=
         and[equal[image[ordlist[x], v], intersection[OMEGA, x, APPLY[ordlist[x], u]]],
          member[u, domain[ordlist[x]]], member[v, V]]
```

The required membership rule is:

```
In[12]:= (member[y, fix[z]] // AssertTest) /. z -> 
          composite[inverse[IMAGE[ordlist[x]]], IMAGE[id[intersection[OMEGA, x]]], ordlist[x]]

Out[12]= member[y, fix[composite[inverse[IMAGE[ordlist[x]]],
           IMAGE[id[intersection[OMEGA, x]]], ordlist[x]]]] ==
         and[equal[image[ordlist[x], y], intersection[OMEGA, x, APPLY[ordlist[x], y]]],
          member[y, domain[ordlist[x]]]]
```

```
In[13]:= member[y_, fix[composite[inverse[IMAGE[ordlist[x_]]],
           IMAGE[id[intersection[OMEGA, x_]]], ordlist[x_]]]] :=
         and[equal[image[ordlist[x], y], intersection[OMEGA, x, APPLY[ordlist[x], y]]],
          member[y, domain[ordlist[x]]]]
```

---

the set of ordinals in x less than a listed ordinal

Lemma.

```
In[14]:= Map[not,
          SubstTest[and, implies[and[p1, r1], r3], implies[and[q1, r1, r2, r3], or[r4, r5]],
           implies[and[r3, r5], r6], implies[and[r1, r6], r7], not[and[r2, r3, r7]],
           implies[and[q1, r3, r4], r8], not[implies[and[p1, q1, r1, r2], r4]],
           {p1 -> subclass[x, OMEGA], q1 → member[y, OMEGA],
            r1 → member[v, x], r2 -> member[v, hull[x, succ[y]]], r3 → member[v, OMEGA],
            r4 → subclass[v, y], r5 → member[y, v], r6 → subclass[succ[y], v],
            r7 → subclass[hull[x, succ[y]], v], r8 → member[v, succ[y]]}]]

Out[14]= or[not[member[v, x]], not[member[v, hull[x, succ[y]]]],
           not[member[y, OMEGA]], not[subclass[x, OMEGA]], subclass[v, y]] == True
```

```
In[15]:= (% /. {x → x_, y → y_, v → v_}) /. Equal → SetDelayed
```

Eliminating the set variable **v** yields:

```
In[16]:= Map[equal[V, #] &,
           SubstTest[class, v, or[not[member[v, x]], not[member[v, u]], not[member[y, w]],
             not[subclass[x, w]], subclass[v, y]], {u → hull[x, succ[y]], w → OMEGA}]] // Reverse

Out[16]= or[not[member[y, OMEGA]], not[subclass[x, OMEGA]],
           subclass[U[intersection[x, hull[x, succ[y]]]], y]] == True
```

```
In[17]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Specializing to the case of listing ordinals, one finds:

```
In[18]:= SubstTest[implies, and[member[v, OMEGA], subclass[u, OMEGA]],
          subclass[U[intersection[u, hull[u, succ[v]]]], v],
          {u → intersection[OMEGA, x], v :> APPLY[ordlist[x], y]}]

Out[18]= or[not[member[y, domain[ordlist[x]]]], subclass[
           U[intersection[OMEGA, x, APPLY[ordlist[x], succ[y]]]], APPLY[ordlist[x], y]]] == True

In[19]:= or[not[member[y_, domain[ordlist[x_]]]],
          subclass[U[intersection[OMEGA, x_, APPLY[ordlist[x_], succ[y_]]]],
           APPLY[ordlist[x_], y_]]] := True
```

One needs to make a special effort to deal with the sum class constructor **U** in this statement. Note that any statement of the form **subclass[U[x], y]** is equivalent to one of the form **subclass[x, P[y]]**. The following temporary rewrite rule shows how the power class can be replaced with a successor in the present situation.

```
In[20]:= SubstTest[implies, equal[x, ord[y]],
          equal[intersection[OMEGA, P[x]], intersection[OMEGA, succ[x]]], y → x]

Out[20]= or[equal[intersection[OMEGA, P[x]], intersection[OMEGA, succ[x]]],
           not[member[x, OMEGA]]] == True

In[21]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[22]:= implies[or[equal[V, z], member[z, OMEGA]],
           equal[intersection[OMEGA, P[z]], intersection[OMEGA, succ[z]]]] // NotNotTest

Out[22]= or[and[not[equal[V, z]], not[member[z, OMEGA]]],
           equal[intersection[OMEGA, P[z]], intersection[OMEGA, succ[z]]]] == True

In[23]:= (% /. z → z_) /. Equal → SetDelayed
```

In general, **APPLY[ordlist[x], y]** is either an ordinal or else is equal to **V**. Applying the above lemma, one finds:

```
In[24]:= SubstTest[implies, or[equal[V, z], member[z, OMEGA]],
          equal[intersection[OMEGA, P[z]], intersection[OMEGA, succ[z]]],
          z :> APPLY[ordlist[x], y]]

Out[24]= equal[intersection[OMEGA, P[APPLY[ordlist[x], y]]],
           intersection[OMEGA, succ[APPLY[ordlist[x], y]]]] == True

In[25]:= intersection[OMEGA, P[APPLY[ordlist[x_], y_]]] :=
          intersection[OMEGA, succ[APPLY[ordlist[x], y]]]
```

This allows one to reformulate the inclusion derived above so that **U** is eliminated in terms of successors:

```
In[26]:= Map[implies[member[y, domain[ordlist[x]]], #] &,
          SubstTest[subclass, u, intersection[v, w],
           {u -> intersection[OMEGA, x, APPLY[ordlist[x], succ[y]]],
            v → OMEGA, w → P[APPLY[ordlist[x], y]]}]]

Out[26]= or[not[member[y, domain[ordlist[x]]]],
           subclass[intersection[OMEGA, x, APPLY[ordlist[x], succ[y]]],
            succ[APPLY[ordlist[x], y]]] == True

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

Lemma.

```
In[28]:= SubstTest[implies, equal[v, V], subclass[z, succ[v]], v → APPLY[ordlist[x], y]]

Out[28]= or[member[y, domain[ordlist[x]]], subclass[z, succ[APPLY[ordlist[x], y]]] == True

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

The condition that **y** be a member of the domain can be omitted:

```
In[30]:= SubstTest[and, implies[p1, p2], or[p1, p2],
           {p1 -> member[y, domain[ordlist[x]]], p2 -> subclass[intersection[OMEGA, x,
               APPLY[ordlist[x], succ[y]]], succ[APPLY[ordlist[x], y]]]}] // Reverse

Out[30]= subclass[intersection[OMEGA, x, APPLY[ordlist[x], succ[y]]],
           succ[APPLY[ordlist[x], y]]] == True

In[31]:= subclass[intersection[OMEGA, x_, APPLY[ordlist[x_], succ[y_]]],
           succ[APPLY[ordlist[x_], y_]]] := True
```

Lemma.

```
In[32]:= SubstTest[subclass, union[u, v], w,
          {u -> APPLY[ordlist[x], y], v -> set[APPLY[ordlist[x], y]],
           w → union[APPLY[ordlist[x], succ[y]], complement[OMEGA], complement[x]]}]

Out[32]= subclass[intersection[OMEGA, x, succ[APPLY[ordlist[x], y]]],
           APPLY[ordlist[x], succ[y]]] == True

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

Theorem.

```
In[34]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> intersection[OMEGA, x, APPLY[ordlist[x], succ[y]]],
           v -> intersection[OMEGA, x, succ[APPLY[ordlist[x], y]]]}]

Out[34]= True == equal[intersection[OMEGA, x, APPLY[ordlist[x], succ[y]]],
           intersection[OMEGA, x, succ[APPLY[ordlist[x], y]]]]

In[35]:= intersection[OMEGA, x_, succ[APPLY[ordlist[x_], y_]]] :=
          intersection[OMEGA, x, APPLY[ordlist[x], succ[y]]]
```

Temporary lemma.

```
In[36]:= SubstTest[image, u, union[v, w], {u → ordlist[x], v → y, w → set[y]}]

Out[36]= image[ordlist[x], succ[y]] == union[image[ordlist[x], y], set[APPLY[ordlist[x], y]]]

In[37]:= image[ordlist[x_], succ[y_]] := union[image[ordlist[x], y], set[APPLY[ordlist[x], y]]]
```

Lemma:

```
In[38]:= equal[intersection[OMEGA, x, set[APPLY[ordlist[x], y]]], set[APPLY[ordlist[x], y]]]

Out[38]= True

In[39]:= intersection[OMEGA, x_, set[APPLY[ordlist[x_], y_]]] := set[APPLY[ordlist[x], y]]
```

The main result of this section compares the set of ordinals in **x** less than a particular listed ordinal **w = APPLY[ordlist[x], y]** with the set of ordinals in **x** less than the next ordinal **APPLY[ordlist[x], succ[y]]** on the list. One finds that only one new member has been added, namely **w**.

```
In[40]:= SubstTest[intersection, u, union[v, w],
          {u → intersection[OMEGA, x], v -> APPLY[ordlist[x], y], w → set[APPLY[ordlist[x], y]]}]

Out[40]= intersection[OMEGA, x, APPLY[ordlist[x], succ[y]]] ==
          union[intersection[OMEGA, x, APPLY[ordlist[x], y]], set[APPLY[ordlist[x], y]]]

In[41]:= intersection[OMEGA, x_, APPLY[ordlist[x_], succ[y_]]] :=
          union[intersection[OMEGA, x, APPLY[ordlist[x], y]], set[APPLY[ordlist[x], y]]]
```

---

induction step

It is convenient to introduce a temporary abbreviation **ind[x]** for the class to which induction will be applied.

```
In[42]:= ind[x_] := union[fix[composite[inverse[IMAGE[ordlist[x]]],
             IMAGE[id[intersection[OMEGA, x]]], ordlist[x]]], complement[domain[ordlist[x]]]]
```

The results of the preceding section state that **ind[x]** is successor-invariant:

```
In[43]:= implies[member[y, ind[x]], member[succ[y], ind[x]]] // NotNotTest

Out[43]= or[and[member[y, domain[ordlist[x]]],
            not[equal[image[ordlist[x], y], intersection[OMEGA, x, APPLY[ordlist[x], y]]]]],
           equal[union[image[ordlist[x], y], set[APPLY[ordlist[x], y]]],
            union[intersection[OMEGA, x, APPLY[ordlist[x], y]], set[APPLY[ordlist[x], y]]]],
           not[member[y, V]], not[member[succ[y], domain[ordlist[x]]]]] == True

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

The variable **y** needs to be eliminated before one can apply induction. Various rewrite rules are in place that make this elimination process a little more involved than one might otherwise expect. One could of course temporarily remove these rules, but that is not necessary. The first step is to show that **ind[x]** is subvariant under **inverse[SUCC]**.

```
In[45]:= member[y, dif[ind[x], image[inverse[SUCC], ind[x]]]] // NotNotTest

Out[45]= or[and[equal[image[ordlist[x], y], intersection[OMEGA, x, APPLY[ordlist[x], y]]],
           member[y, U[domain[ordlist[x]]]],
           not[equal[union[image[ordlist[x], y], set[APPLY[ordlist[x], y]]],
             union[intersection[OMEGA, x, APPLY[ordlist[x], y]], set[APPLY[ordlist[x], y]]]]],
         and[equal[image[ordlist[x], y], intersection[OMEGA, x, APPLY[ordlist[x], y]]],
           member[y, U[domain[ordlist[x]]]], not[member[succ[y], domain[ordlist[x]]]]]] == False

In[46]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The variable **y** is now eliminated by using **Normality**.

```
In[47]:= Map[equal[0, #] &, dif[ind[x], image[inverse[SUCC], ind[x]]] // Normality]

Out[47]= subclass[image[SUCC, intersection[fix[composite[inverse[IMAGE[ordlist[x]]],
             IMAGE[id[intersection[OMEGA, x]]], ordlist[x]]], U[domain[ordlist[x]]]]],
           fix[composite[inverse[IMAGE[ordlist[x]]], IMAGE[id[intersection[OMEGA, x]]],
             ordlist[x]]]] == True

In[48]:= (% /. x → x_) /. Equal → SetDelayed
```

The conversion from subvariance to invariance is done as follows:

```
In[49]:= SubstTest[subvariant, inverse[SUCC], y, y → ind[x]] // Reverse

Out[49]= subclass[intersection[domain[ordlist[x]],
            image[SUCC, fix[composite[inverse[IMAGE[ordlist[x]]],
              IMAGE[id[intersection[OMEGA, x]]], ordlist[x]]]]],
           fix[composite[inverse[IMAGE[ordlist[x]]], IMAGE[id[intersection[OMEGA, x]]],
             ordlist[x]]]] == True

In[50]:= (% /. x → x_) /. Equal → SetDelayed
```

The induction step is:

```
In[51]:= SubstTest[implies, INDUCTIVE[z], subclass[omega, z], z → ind[x]]

Out[51]= subclass[domain[ordlist[x]], fix[composite[inverse[IMAGE[ordlist[x]]],
             IMAGE[id[intersection[OMEGA, x]]], ordlist[x]]]] == True

In[52]:= (% /. x → x_) /. Equal → SetDelayed
```

This inclusion can be strengthened to an equation:

```
In[53]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u -> domain[ordlist[x]], v -> fix[composite[inverse[IMAGE[ordlist[x]]],
              IMAGE[id[intersection[OMEGA, x]]], ordlist[x]]]}] // Reverse

Out[53]= equal[domain[ordlist[x]], fix[composite[inverse[IMAGE[ordlist[x]]],
           IMAGE[id[intersection[OMEGA, x]]], ordlist[x]]]] == True

In[54]:= fix[composite[inverse[IMAGE[ordlist[x_]]],
            IMAGE[id[intersection[OMEGA, x_]]], ordlist[x_]]] := domain[ordlist[x]]
```

Main theorem.

```
In[55]:= Map[implies[member[y, domain[ordlist[x]]], #] &,
           (member[y, fix[z]] // AssertTest) /. z -> composite[inverse[IMAGE[ordlist[x]]],
              IMAGE[id[intersection[OMEGA, x]]], ordlist[x]]] // Reverse

Out[55]= or[equal[image[ordlist[x], y], intersection[OMEGA, x, APPLY[ordlist[x], y]]],
           not[member[y, domain[ordlist[x]]]]] == True

In[56]:= or[equal[image[ordlist[x_], y_], intersection[OMEGA, x_, APPLY[ordlist[x_], y_]]],
           not[member[y_, domain[ordlist[x_]]]]] := True
```

eliminating the variable y

In this section, a version of the no-gap theorem is derived that involves only the variable **x**. The first step yields an inclusion:

```
In[57]:= Map[equal[0, #] &, SubstTest[composite, w, id[domain[w]],
            w → dif[composite[IMAGE[id[intersection[OMEGA, x]]], ordlist[x]],
              IMAGE[ordlist[x]]]] // Reverse]

Out[57]= subclass[
           composite[IMAGE[id[intersection[OMEGA, x]]], ordlist[x]], IMAGE[ordlist[x]]] == True

In[58]:= (% /. x → x_) /. Equal → SetDelayed
```

When one function is a subclass of another, one can rewrite this as an equation stating that the one is a restriction of the other:

```
In[59]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
          equal[u, composite[v, id[domain[u]]]],
          {u -> composite[IMAGE[id[intersection[OMEGA, x]]], ordlist[x]],
           v -> IMAGE[ordlist[x]]}]

Out[59]= equal[composite[IMAGE[id[intersection[OMEGA, x]]], ordlist[x]],
           composite[IMAGE[ordlist[x]], id[domain[ordlist[x]]]]] == True

In[60]:= composite[IMAGE[id[intersection[OMEGA, x_]]], ordlist[x_]] :=
          composite[IMAGE[ordlist[x]], id[domain[ordlist[x]]]]
```

a corollary about range[ordlist[x]]

If one throws away the information about the order in which ordinals are listed, and just concentrates on which ordinals are listed, one obtains a corollary that restates the no-gap condition as a property of **range[ordlist[x]]**. This result is most easily obtained from the equation derived in the preceding section:

```
In[61]:= Map[subclass[U[#], range[ordlist[x]]] &,
           ImageComp[IMAGE[id[intersection[OMEGA, x]]], ordlist[x], V]] // Reverse

Out[61]= subclass[intersection[OMEGA, x, U[range[ordlist[x]]]], range[ordlist[x]]] == True

In[62]:= subclass[intersection[OMEGA, x_, U[range[ordlist[x_]]]], range[ordlist[x_]]] := True
```

Introducing a variable helps to understand this corollary.

```
In[63]:= SubstTest[implies, and[member[w, u], subclass[u, v]], member[w, v],
           {u -> range[ordlist[x]],
            v → P[union[complement[OMEGA], complement[x], range[ordlist[x]]]]}]

Out[63]= or[not[member[w, range[ordlist[x]]]],
           subclass[intersection[OMEGA, w, x], range[ordlist[x]]] == True

In[64]:= or[not[member[w_, range[ordlist[x_]]]],
           subclass[intersection[OMEGA, w_, x_], range[ordlist[x_]]] := True
```

This says that if **w** is an ordinal that is listed by **ordlist[x]**, then all ordinals in **x** less than **w** are also listed.