enumerating all ordinals in a given class: part 5. examples

Johan G. F. Belinfante
2010 October 8

```
In[1]:= SetDirectory["l:"]; << goedel.10oct07a

        :Package Title: goedel.10oct07a            2010 October 7 at 2:25 p.m.

        It is now:  2010 Oct 8 at 15:45

        Loading Simplification Rules

        TOOLS.M is now incorporated in the GOEDEL program as of 2010 September 3

        weightlimit = 40
```

summary

In this fifth notebook about enumerating ordinals some examples are presented. A key observation is that if a partial enumeration lists all ordinals in a class, then it is the complete enumeration.

equations involving inverses

The **GOEDEL** program has this basic simplification rule concerning equations involving inverses.

```
In[2]:= equal[inverse[x], inverse[y]]

Out[2]= equal[composite[Id, x], composite[Id, y]]
```

In this section, two related results of a rather similar nature are derived.

Theorem. A simplification rule that eliminates two occurrences of **inverse**.

```
In[3]:= SubstTest[equal, inverse[t], inverse[z], t → composite[y, id[x]]] // Reverse

Out[3]= equal[composite[id[x], inverse[y]], inverse[z]] ==
         equal[composite[Id, z], composite[y, id[x]]]

In[4]:= equal[composite[id[x_], inverse[y_]], inverse[z_]] :=
         equal[composite[Id, z], composite[y, id[x]]]
```

Theorem. A similar simplification rule.

```
In[5]:= SubstTest[equal, inverse[t], inverse[z], t → composite[id[y], x]] // Reverse

Out[5]= equal[composite[inverse[x], id[y]], inverse[z]] ==
         equal[composite[Id, z], composite[id[y], x]]

In[6]:= equal[composite[inverse[x_], id[y_]], inverse[z_]] :=
         equal[composite[Id, z], composite[id[y], x]]
```

subclasses of one-to-one functions

Any subclass of a function is a restriction. The **GOEDEL** program has this basic rewrite rule concerning this fact.

```
In[7]:= equal[x, composite[funpart[y], id[domain[x]]]]

Out[7]= subclass[x, funpart[y]]
```

In this section a similar rule is derived for one-to-one functions, with **domain** replaced by **range**.

Lemma. A temporary rewrite rule obtained from the existing rule about domains.

```
In[8]:= SubstTest[equal, t, composite[funpart[v], id[domain[t]]],
          {t → inverse[x], v → inverse[oopart[y]]}] // Reverse

Out[8]= equal[composite[Id, x], composite[id[range[x]], oopart[y]]] ==
         subclass[composite[Id, x], oopart[y]]

In[9]:= equal[composite[Id, x_], composite[id[range[x_]], oopart[y_]]] :=
         subclass[composite[Id, x], oopart[y]]
```

In the derivation of the following lemma, a temporary variable **t** is introduced, and then eliminated at the end. This trick blocks undesirable rewriting that would otherwise take place.

Lemma.

```
In[10]:= (Map[not, SubstTest[and, implies[p1, p2],
              implies[and[p0, p2], p3], implies[and[p0, p1], p4], implies[and[p3, p4], p5],
              not[implies[and[p0, p1], p5]], {p0 → equal[t, composite[Id, x]],
               p1 → subclass[x, oopart[y]], p2 → subclass[x, cart[V, V]], p3 → equal[x, t],
               p4 -> equal[t, composite[id[range[x]], oopart[y]]], p5 -> equal[x,
                 composite[id[range[x]], oopart[y]]]}]] // Reverse) /. t → composite[Id, x]

Out[10]= or[equal[x, composite[id[range[x]], oopart[y]]], not[subclass[x, oopart[y]]]] == True

In[11]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. Any subclass of a one-to-one function is a co-restriction.

```
In[12]:= equiv[equal[x, composite[id[range[x]], oopart[y]]], subclass[x, oopart[y]]]

Out[12]= True
```

```
In[13]:= equal[x_, composite[id[range[x_]], oopart[y_]]] := subclass[x, oopart[y]]
```

partial enumerations that list all ordinals

A key observation is derived in this section that shortens the derivations of most of the remaining results obtained in this notebook.

Lemma.

```
In[14]:= SubstTest[implies, subclass[u, v], subclass[range[u], range[v]],
           {u → enum[x], v → composite[HULL[intersection[x, OMEGA]], IMAGE[enum[x]]]}] // Reverse
Out[14]= subclass[range[enum[x]], image[HULL[intersection[OMEGA, x]], P[range[enum[x]]]]] == True

In[15]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. An upper bound for **range[enum[x]]**.

```
In[16]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u -> range[enum[x]], v -> image[HULL[intersection[OMEGA, x]], P[range[enum[x]]]],
            w → range[HULL[intersection[OMEGA, x]]]}] // Reverse
Out[16]= subclass[range[enum[x]], x] == True

In[17]:= subclass[range[enum[x_]], x_] := True
```

At this point one has the inclusion **range[enum[x]]** $\subset \Omega \bigcap$ **x**. The reverse inclusion has not been established and will not be needed in this notebook.

Lemma. A temporary simplification rule.

```
In[18]:= SubstTest[subclass, composite[Id, t],
           cart[V, y], {t → enum[x], y → intersection[OMEGA, x]}] // Reverse
Out[18]= subclass[enum[x], cart[V, intersection[OMEGA, x]]] == True

In[19]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. A simplification rule.

```
In[20]:= equal[composite[id[intersection[OMEGA, x]], enum[x]], enum[x]]
Out[20]= True

In[21]:= composite[id[intersection[OMEGA, x_]], enum[x_]] := enum[x]
```

Lemma. Any partial enumeration is a co-restriction of **enum[x]**.

```
In[22]:= Map[implies[member[w, partenum[x]], #] &,
           SubstTest[equal, w, composite[id[range[w]], oopart[t]], t → enum[x]]] // Reverse

Out[22]= or[equal[w, composite[id[range[w]], enum[x]]], not[member[w, partenum[x]]]] == True

In[23]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Theorem. If a partial enumeration already lists all ordinals in a class, then it is the complete enumeration.

```
In[24]:= SubstTest[implies, and[equal[t, range[w]], member[w, partenum[x]]],
           equal[w, composite[id[t], enum[x]]], t → intersection[OMEGA, x]] // Reverse

Out[24]= or[equal[w, enum[x]],
           not[equal[intersection[OMEGA, x], range[w]]], not[member[w, partenum[x]]]] == True

In[25]:= or[equal[w_, enum[x_]], not[equal[intersection[OMEGA, x_], range[w_]]],
           not[member[w_, partenum[x_]]]] := True
```

An easy application of this result concerns the function that enumerates all primes.

Lemma. The prime sequence is a partial enumeration for the set of primes.

```
In[26]:= SubstTest[and, member[domain[w], OMEGA],
           subclass[w, composite[HULL[intersection[OMEGA, x]], IMAGE[w]]],
           {w → PRIMESEQ, x → PRIMES}]

Out[26]= member[PRIMESEQ, partenum[PRIMES]] == True

In[27]:= member[PRIMESEQ, partenum[PRIMES]] := True
```

Theorem. The prime sequence **PRIMESEQ** enumerates all ordinals in the class of primes in increasing order.

```
In[28]:= SubstTest[implies,
           and[member[w, partenum[t]], equal[intersection[OMEGA, t], range[w]]],
           equal[w, enum[t]], {t → PRIMES, w → PRIMESEQ}] // Reverse

Out[28]= equal[PRIMESEQ, enum[PRIMES]] == True

In[29]:= enum[PRIMES] := PRIMESEQ
```

enumerating the ordinals in ord[x] and in Ω

In this section, explicit formulas are derived for **enum[ord[x]]** and **enum[Ω]**.

Lemma. The identity function restricted to any ordinal is a partial enumeration of that ordinal.

```
In[30]:= SubstTest[and, member[domain[w], OMEGA],
           subclass[w, composite[HULL[intersection[OMEGA, t]], IMAGE[w]]],
           {w → id[ord[x]], t → ord[x]}]

Out[30]= member[id[ord[x]], partenum[ord[x]]] == True
```

```
In[31]:= member[id[ord[x_]], partenum[ord[x_]]] := True
```

Theorem. The function **enum[ord[x]]** is the identity on **ord[x]**.

```
In[32]:= SubstTest[implies,
           and[member[w, partenum[t]], equal[intersection[OMEGA, t], range[w]]],
           equal[w, enum[t]], {t → ord[x], w → id[ord[x]]}] // Reverse
```

```
Out[32]= equal[enum[ord[x]], id[ord[x]]] == True
```

```
In[33]:= enum[ord[x_]] := id[ord[x]]
```

Corollary. (Removing the **ord** wrapper.)

```
In[34]:= SubstTest[implies, equal[x, ord[t]], equal[enum[x], id[x]], t → x] // Reverse
```

```
Out[34]= or[equal[enum[x], id[x]], not[member[x, OMEGA]]] == True
```

```
In[35]:= or[equal[enum[x_], id[x_]], not[member[x_, OMEGA]]] := True
```

A similar result holds when **x** is the class of all ordinals. The technique used for ordinals fails when listing all ordinals because **enum[Ω]** is a proper class, and hence is not a member of **partenum[Ω]**. But it is easy enough to do this special case by a different method.

Lemma. Simplification rule.

```
In[36]:= Assoc[id[P[OMEGA]], id[P[ord[x]]], IMAGE[id[ord[x]]]]
```

```
Out[36]= composite[id[P[OMEGA]], IMAGE[id[ord[x]]]] == IMAGE[id[ord[x]]]
```

```
In[37]:= composite[id[P[OMEGA]], IMAGE[id[ord[x_]]]] := IMAGE[id[ord[x]]]
```

Theorem. The identity relation restricted to any ordinal is a partial enumeration of the class of all ordinals.

```
In[38]:= SubstTest[and, member[domain[w], OMEGA],
          subclass[w, composite[HULL[intersection[OMEGA, t]], IMAGE[w]]],
          {w → id[ord[x]], t → OMEGA}]
```

```
Out[38]= member[id[ord[x]], partenum[OMEGA]] == True
```

```
In[39]:= member[id[ord[x_]], partenum[OMEGA]] := True
```

The variable **x** can now be eliminated using **reify**.

Theorem. The class of all identity relations restricted to ordinals is a subclass of the class or partial enumerations of **Ω**.

```
In[40]:= Map[empty, SubstTest[reify, x, dif[set[id[ord[x]]], t], t -> partenum[OMEGA]]]
```

```
Out[40]= subclass[image[IMAGE[DUP], OMEGA], partenum[OMEGA]] == True
```

```
In[41]:= % /. Equal→ SetDelayed
```

Corollary.

```
In[42]:= SubstTest[implies, subclass[u, v], subclass[U[u], U[v]],
           {u -> image[IMAGE[DUP], OMEGA], v -> partenum[OMEGA]}] // Reverse

Out[42]= subclass[OMEGA, fix[enum[OMEGA]]] == True

In[43]:= % /. Equal → SetDelayed
```

Theorem. The function that enumerates all ordinals in order is the identity relation on the class of ordinals.

```
In[44]:= SubstTest[equal, u, composite[funpart[t], id[domain[u]]],
           {u → id[OMEGA], t → enum[OMEGA]}] // Reverse

Out[44]= equal[enum[OMEGA], id[OMEGA]] == True

In[45]:= enum[OMEGA] := id[OMEGA]
```

Corollary. An explicit formula for the function **enum[V]**.

```
In[46]:= SubstTest[enum, intersection[OMEGA, x], x → V]

Out[46]= enum[V] == id[OMEGA]

In[47]:= enum[V] := id[OMEGA]
```

Theorem. An explicit formula for the class of partial enumerations of **Ω**.

```
In[48]:= SubstTest[image, IMAGE[composite[id[enum[x]], inverse[FIRST]]], OMEGA, x → OMEGA]

Out[48]= partenum[OMEGA] == image[IMAGE[DUP], OMEGA]

In[49]:= partenum[OMEGA] := image[IMAGE[DUP], OMEGA]
```

Corollary.

```
In[50]:= SubstTest[partenum, intersection[OMEGA, x], x → V]

Out[50]= partenum[V] == image[IMAGE[DUP], OMEGA]

In[51]:= partenum[V] := image[IMAGE[DUP], OMEGA]
```

relating ordlist[x] to enum[x]

The function **ordlist[x]** that lists in order the first countably many ordinals in a class **x** was defined using iteration:

```
In[52]:= iterate[composite[HULL[intersection[OMEGA, x]], SUCC], set[A[intersection[OMEGA, x]]]]

Out[52]= ordlist[x]
```

This section is concerned with the relation between the functions **ordlist[x]** and **enum[x]**.

Theorem. The function **ordlist[x]** is a partial enumeration of the ordinals in **x**.

```
In[53]:= SubstTest[and, member[domain[w], OMEGA],
          subclass[w, composite[HULL[intersection[OMEGA, x]], IMAGE[w]]], w → ordlist[x]]

Out[53]= member[ordlist[x], partenum[x]] == True

In[54]:= member[ordlist[x_], partenum[x_]] := True
```

Corollary. The function **ordlist[x]** is a restriction of **enum[x]**.

```
In[55]:= SubstTest[implies, member[w, partenum[x]],
            equal[w, composite[enum[x], id[domain[w]]]], w → ordlist[x]] // Reverse

Out[55]= equal[composite[enum[x], id[domain[ordlist[x]]]], ordlist[x]] == True

In[56]:= composite[enum[x_], id[domain[ordlist[x_]]]] := ordlist[x]
```

It will now be shown that the equation **ordlist[x] = enum[x] ∘ id[ω]** holds. This will be done by considering separately the case that **x** holds only a finite number of ordinals and the case that **x** holds infinitely many ordinals. The second case will be considered first.

Lemma. The case that **Ω ⋂ x** is infinite.

```
In[57]:= SubstTest[implies, equal[u, v], equal[composite[t, id[u]], composite[t, id[v]]],
            {t → enum[x], u → omega, v → domain[ordlist[x]]}] // Reverse

Out[57]= or[equal[composite[enum[x], id[omega]], ordlist[x]],
            member[intersection[OMEGA, x], FINITE]] == True

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

If **x** holds only a fnite number of ordinals, then **ordlist[x]** enumerates them all, and in that case **ordlist[x] = enum[x]**.

Lemma.

```
In[59]:= SubstTest[implies,
            and[member[w, partenum[x]], equal[intersection[OMEGA, x], range[w]]],
            equal[w, enum[x]], w → ordlist[x]] // Reverse

Out[59]= or[equal[enum[x], ordlist[x]],
            not[equal[intersection[OMEGA, x], range[ordlist[x]]]]] == True

In[60]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. The case that **Ω ⋂ x** is finite.

```
In[61]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
             not[implies[p1, p3]], {p1 → member[intersection[OMEGA, x], FINITE],
              p2 -> equal[intersection[OMEGA, x], range[ordlist[x]]],
              p3 -> equal[enum[x], ordlist[x]]}]] // Reverse

Out[61]= or[equal[enum[x], ordlist[x]], not[member[intersection[OMEGA, x], FINITE]]] == True

In[62]:= or[equal[enum[x_], ordlist[x_]], not[member[intersection[OMEGA, x_], FINITE]]] := True
```

Theorem. The function **ordlist[x]** is the restriction of **enum[x]** to the set ω of natural numbers.

```
In[63]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], or[p1, p3], not[p3],
            {p1 -> member[intersection[OMEGA, x], FINITE], p2 -> equal[enum[x], ordlist[x]],
             p3 → equal[composite[enum[x], id[omega]], ordlist[x]]}]] // Reverse

Out[63]= equal[composite[enum[x], id[omega]], ordlist[x]] == True

In[64]:= composite[enum[x_], id[omega]] := ordlist[x]
```