enumerating ordinals in a given class, part 7. adding a new point to a partial enumeration

Johan G. F. Belinfante
2010 October 14

```
In[1]:= SetDirectory["l:"]; << goedel.10oct12a

        :Package Title: goedel.10oct12a            2010 October 12 at 8:10 a.m.

        It is now:  2010 Oct 14 at 16:18

        Loading Simplification Rules

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

        weightlimit = 40
```

summary

In this seventh notebook in a series about enumerating ordinals in a given class the focus is about extending a partial enumeration by adding a single new point. It was shown in an earlier notebook that the following recursion equation holds for any member of the domain of **enum[x]**.

```
In[2]:= implies[member[y, domain[enum[x]]],
          equal[APPLY[enum[x], y], hull[intersection[OMEGA, x], image[enum[x], y]]]]

Out[2]= True
```

In this notebook it is shown that the same recursion equation holds when **image[enum[x], y]** is not a subset of an ordinal belonging to **x**, and also for any ordinal contained in the domain of **enum[x]**.

a trivial case

The recursion equation is trivially satisfied when both sides of the equation are equal to **V**. The following lemma considers this case, but it contains a redundant literal.

Lemma.

```
In[3]:= SubstTest[implies, and[equal[u, v], equal[v, w]], equal[u, w], {u -> APPLY[enum[x], y],
            v → V, w -> hull[intersection[OMEGA, x], image[enum[x], y]]}] // Reverse

Out[3]= or[equal[APPLY[enum[x], y], hull[intersection[OMEGA, x], image[enum[x], y]]],
          member[y, domain[enum[x]]],
          member[image[enum[x], y], image[inverse[S], intersection[OMEGA, x]]]] == True
```

```
In[4]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The following result is obtained by eliminating the redundant second literal in the lemma.

Theorem. The recursion equation holds when **image[enum[x], y]** is not a subset of an ordinal that belongs to **x**.

```
In[5]:= SubstTest[and, implies[p, q], or[p, q], {p → member[y, domain[enum[x]]],
          q -> or[equal[APPLY[enum[x], y], hull[intersection[OMEGA, x], image[enum[x], y]]],
            member[image[enum[x], y], image[inverse[S], intersection[OMEGA, x]]]]}]

Out[5]= or[equal[APPLY[enum[x], y], hull[intersection[OMEGA, x], image[enum[x], y]]],
          member[image[enum[x], y], image[inverse[S], intersection[OMEGA, x]]] == True

In[6]:= or[equal[APPLY[enum[x_], y_], hull[intersection[OMEGA, x_], image[enum[x_], y_]]],
          member[image[enum[x_], y_], image[inverse[S], intersection[OMEGA, x_]]] := True
```

extending a partial enumeration

This section consists mainly of some observations that help motivate the derivations in the next section. Recall that the restriction of **enum[x]** to any ordinal is a partial enumeration.

```
In[7]:= member[composite[enum[x], id[ord[y]]], partenum[x]]

Out[7]= True
```

The same of course holds for the next ordinal:

```
In[8]:= SubstTest[member, composite[enum[x], id[ord[t]]],
          partenum[x], t → succ[ord[y]]] // Reverse

Out[8]= member[composite[enum[x], id[succ[ord[y]]]], partenum[x]] == True
```

Theorem. The restriction of **enum[x]** to **succ[ord[y]]** is the union of the restriction of **enum[x]** to **ord[y]** and possibly one new point **pair[ord[y], APPLY[enum[x], ord[y]]]**.

```
In[9]:= SubstTest[composite, enum[x], union[u, v], {u → id[ord[y]], v → id[set[ord[y]]]}]

Out[9]= union[cart[set[ord[y]], set[APPLY[enum[x], ord[y]]]], composite[enum[x], id[ord[y]]]] ==
         composite[enum[x], id[succ[ord[y]]]]

In[10]:= union[cart[set[ord[y_]], set[APPLY[enum[x_], ord[y_]]]],
           composite[enum[x_], id[ord[y_]]]] := composite[enum[x], id[succ[ord[y]]]]
```

If **ord[y] ∈ domain[enum[x]]** a bonafide new point is being added here, but otherwise one is simply adjoining the empty set, and no real extension is being made. In the next section, a similar extension is considered, but the point being added is not assumed to have the correct second coordinate **APPLY[enum[x], ord[y]]**.

the case of an ordinal contained in domain[enum[x]]

In this section it is shown that the recursion equation holds for any ordinal contained in the domain of **enum[x]**. The following result will be used.

Theorem. A consequence of the fact that any partial enumeration is a subset of the total enumeration.

```
In[11]:= or[not[member[domain[t], OMEGA]],
            not[subclass[t, composite[HULL[intersection[OMEGA, x]], IMAGE[t]]]],
            subclass[t, enum[x]]] // NotNotTest

Out[11]= or[not[member[domain[t], OMEGA]],
            not[subclass[t, composite[HULL[intersection[OMEGA, x]], IMAGE[t]]]],
            subclass[t, enum[x]]] == True

In[12]:= or[not[member[domain[t_], OMEGA]],
            not[subclass[t_, composite[HULL[intersection[OMEGA, x_]], IMAGE[t_]]]],
            subclass[t_, enum[x_]]] := True
```

The idea is to apply the above result to the case of the union of the partial enumeration **enum[x] ∘ id[ord[y]]** and the singleton of **PAIR[ord[y], hull[Ω ⋂ x, image[enum[x], y]]]**. This would be the right point to add if the recursion equation were true, but of course this is not being assumed here. Some lemmas are needed to simplify the expressions obtained when one substitutes this union for **t** in the above theorem.

Lemma. A simplification rule.

```
In[13]:= equal[succ[ord[y]],
            union[intersection[image[V, w], set[ord[y]]], ord[y]]] // AssertTest

Out[13]= equal[succ[ord[y]], union[intersection[image[V, w], set[ord[y]]], ord[y]]] ==
          not[equal[0, w]]

In[14]:= equal[succ[ord[y_]], union[intersection[image[V, w_], set[ord[y_]]], ord[y_]]] :=
          not[equal[0, w]]
```

Theorem. An expression which is either **ord[y]** or its successor is an ordinal.

```
In[15]:= (implies[or[equal[t, ord[y]], equal[t, succ[ord[y]]]], member[t, OMEGA]] //
            NotNotTest) /. t -> union[intersection[image[V, x], set[ord[y]]], ord[y]]

Out[15]= member[union[intersection[image[V, x], set[ord[y]]], ord[y]], OMEGA] == True

In[16]:= member[union[intersection[image[V, x_], set[ord[y_]]], ord[y_]], OMEGA] := True
```

Corollary.

```
In[17]:= SubstTest[implies, equal[t, ord[y]],
           member[union[intersection[image[V, x], set[ord[y]]], t], OMEGA],
           t → intersection[ord[y], z]] // Reverse

Out[17]= or[member[union[intersection[z, ord[y]], intersection[image[V, x], set[ord[y]]]],
            OMEGA], not[subclass[ord[y], z]]] == True

In[18]:= or[member[union[intersection[z_, ord[y_]], intersection[image[V, x_], set[ord[y_]]]],
            OMEGA], not[subclass[ord[y_], z_]]] := True
```

Lemma. Simplification rule.

```
In[19]:= equal[intersection[complement[P[complement[set[ord[x]]]]], ord[x]], 0]

Out[19]= True

In[20]:= intersection[complement[P[complement[set[ord[x_]]]]], ord[x_]] := 0
```

Lemma. Simplification rule.

```
In[21]:= SubstTest[composite, IMAGE[id[ord[x]]],
           union[u, v], {u → id[ord[x]], v → id[set[ord[x]]]}] // Reverse

Out[21]= composite[IMAGE[id[ord[x]]], id[succ[ord[x]]]] == id[succ[ord[x]]]

In[22]:= composite[IMAGE[id[ord[x_]]], id[succ[ord[x_]]]] := id[succ[ord[x]]]
```

Lemma.

```
In[23]:= SubstTest[implies, and[subclass[t, u], equal[u, v]], subclass[t, v],
           {u → composite[x, id[z]], v -> composite[y, id[z]]}] // Reverse

Out[23]= or[not[equal[composite[x, id[z]], composite[y, id[z]]]],
           not[subclass[t, x]], not[subclass[t, cart[z, V]]], subclass[t, y]] == True

In[24]:= or[not[equal[composite[x_, id[z_]], composite[y_, id[z_]]]],
           not[subclass[t_, x_]], not[subclass[t_, cart[z_, V]]], subclass[t_, y_]] := True
```

The derivation of the next two theorems are slow presumably because they involve several equality substitutions.

Theorem. The extension satisfies the recursion inclusion.

```
In[25]:= Map[not, SubstTest[and, implies[p0, p1], implies[p0, p2], implies[and[p1, p2], p3],
           not[implies[p0, p3]], {p0 → equal[t, union[composite[enum[x], id[ord[y]]],
              set[PAIR[ord[y], hull[intersection[OMEGA, x], image[enum[x], ord[y]]]]]]],
            p1 → equal[composite[HULL[intersection[OMEGA, x]], IMAGE[t], id[succ[ord[y]]]],
              composite[HULL[intersection[OMEGA, x]], IMAGE[enum[x]], id[succ[ord[y]]]]],
            p2 → subclass[t, composite[HULL[intersection[OMEGA, x]],
              IMAGE[enum[x]], id[succ[ord[y]]]]],
            p3 → subclass[t, composite[HULL[intersection[OMEGA, x]], IMAGE[t]]]}] // Reverse

Out[25]= or[not[equal[t, union[
             cart[set[ord[y]], set[hull[intersection[OMEGA, x], image[enum[x], ord[y]]]]],
             composite[enum[x], id[ord[y]]]]],
           subclass[t, composite[HULL[intersection[OMEGA, x]], IMAGE[t]]] == True

In[26]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. A sufficient condition for the recursion equation to hold. This contains a redundant literal.

```
In[27]:= (Map[not, SubstTest[and, implies[and[p0, p1], p2], implies[and[p0, p1], p3],
             implies[and[p2, p3], p4], not[implies[and[p0, p1], p4]],
             {p0 → equal[t, union[composite[enum[x], id[ord[y]]],
                 set[PAIR[ord[y], hull[intersection[OMEGA, x], image[enum[x], ord[y]]]]]]],
              p1 → subclass[ord[y], domain[enum[x]]], p2 → member[domain[t], OMEGA],
              p3 → subclass[t, composite[HULL[intersection[OMEGA, x]], IMAGE[t]]],
              p4 → subclass[t, enum[x]]}]] // Reverse) /.
          t -> union[composite[enum[x], id[ord[y]]],
            set[PAIR[ord[y], hull[intersection[OMEGA, x], image[enum[x], ord[y]]]]]]

Out[27]= or[equal[APPLY[enum[x], ord[y]], hull[intersection[OMEGA, x], image[enum[x], ord[y]]]],
           not[member[image[enum[x], ord[y]], image[inverse[S], intersection[OMEGA, x]]]],
           not[subclass[ord[y], domain[enum[x]]]] == True

In[28]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The third literal is redundant and can be eliminated using the result of the preceding section.

Theorem.

```
In[29]:= SubstTest[and, implies[p, q], or[p, q],
           {p -> member[image[enum[x], ord[y]], image[inverse[S], intersection[OMEGA, x]]],
            q -> or[equal[APPLY[enum[x], ord[y]], hull[intersection[OMEGA, x],
                image[enum[x], ord[y]]]], not[subclass[ord[y], domain[enum[x]]]]]}]

Out[29]= or[equal[APPLY[enum[x], ord[y]], hull[intersection[OMEGA, x], image[enum[x], ord[y]]]],
           not[subclass[ord[y], domain[enum[x]]]] == True

In[30]:= or[equal[APPLY[enum[x_], ord[y_]],
             hull[intersection[OMEGA, x_], image[enum[x_], ord[y_]]]],
           not[subclass[ord[y_], domain[enum[x_]]]]] := True
```

The **ord** wrapper can be eliminated.

Main Theorem. The recursion equation holds for any ordinal contained in the domain of **enum[x]**.

```
In[31]:= SubstTest[implies, equal[y, ord[t]],
           or[equal[APPLY[enum[x], y], hull[intersection[OMEGA, x], image[enum[x], y]]],
            not[subclass[y, domain[enum[x]]]]], t → y] // Reverse

Out[31]= or[equal[APPLY[enum[x], y], hull[intersection[OMEGA, x], image[enum[x], y]]],
           not[member[y, OMEGA]], not[subclass[y, domain[enum[x]]]]] == True

In[32]:= or[equal[APPLY[enum[x_], y_], hull[intersection[OMEGA, x_], image[enum[x_], y_]]],
           not[member[y_, OMEGA]], not[subclass[y_, domain[enum[x_]]]]] := True
```

two final comments

The following counterexample shows that one cannot omit the condition $\mathbf{y \subset \Omega}$ in the main theorem.

```
In[33]:= or[equal[APPLY[enum[x], y], hull[intersection[OMEGA, x], image[enum[x], y]]],
           not[subclass[y, domain[enum[x]]]]] /. {x → V, y → set[set[0]]}

Out[33]= False
```

The earlier-proved validity of the recursion equation for members of **domain[enum[x]]** can be considered to be a special case of the main theorem because this domain is a full class of ordinals.

```
In[34]:= implies[member[y, domain[enum[x]]], and[member[y, OMEGA], subclass[y, OMEGA]]] // not //
          not

Out[34]= True
```