enumerating all ordinals in a given class: part 6. the inclusion enum[x] ⊂ S

Johan G. F. Belinfante
2010 October 11

```
In[1]:= SetDirectory["l:"]; << goedel.10oct09a

        :Package Title: goedel.10oct09a            2010 October 9 at 12:00 midnight

        It is now:  2010 Oct 11 at 11:57

        Loading Simplification Rules

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

        weightlimit = 40
```

summary

This is the sixth notebook in a series about enumerating the ordinals in a given class in increasing order. Some further monotonicity properties of the function **enum[x]** and the inclusion **enum[x] ⊂ S** are derived here.

simplification rules

Since **enum[x]** is a function, it satisfies various standard simplification rules. Three such rules involving function application are derived in this section.

Theorem. Vertical sections are singletons.

```
In[2]:= SubstTest[image, funpart[t], set[y], t → enum[x]] // Reverse

Out[2]= image[enum[x], set[y]] == set[APPLY[enum[x], y]]

In[3]:= image[enum[x_], set[y_]] := set[APPLY[enum[x], y]]
```

Theorem. A membership simplification rule.

```
In[4]:= SubstTest[member, pair[y, APPLY[funpart[t], y]], funpart[t], t → enum[x]] // Reverse

Out[4]= member[pair[y, APPLY[enum[x], y]], enum[x]] == member[y, domain[enum[x]]]

In[5]:= member[pair[y_, APPLY[enum[x_], y_]], enum[x_]] := member[y, domain[enum[x]]]
```

Theorem. Inverse image rule.

```
In[6]:= (member[y, image[inverse[funpart[w]], z]] // AssertTest) /. w → enum[x]

Out[6]= member[y, image[inverse[enum[x]], z]] == member[APPLY[enum[x], y], z]

In[7]:= member[y_, image[inverse[enum[x_]], z_]] := member[APPLY[enum[x], y], z]
```

monotonicity

Any strictly monotone function is monotone in the following sense:

```
In[8]:= subclass[intersection[FUNS, monotone[PS, PS]], monotone[S, S]]

Out[8]= True
```

For functions involving ordinals, the condition of strict monotonicity can be formulated in a variety of ways. One can interchangeably use any of the three relations **E**, **complement[S]** and **PS**.

Lemma. One form of strict monotonicity: **P[enum[x]] ⊂ monotone[E, E]**.

```
In[9]:= Map[subclass[P[enum[x]], #] &,
          SubstTest[intersection, monotone[restrict[x, u, u], restrict[y, v, v]], P[cart[u, v]],
           {u → OMEGA, v → OMEGA, x → complement[S], y → complement[S]}]] // Reverse

Out[9]= subclass[composite[enum[x], inverse[IMAGE[enum[x]]]], S] == True

In[10]:= subclass[composite[enum[x_], inverse[IMAGE[enum[x_]]]], S] := True
```

Theorem. Another form of strict monotonicity: **P[enum[x]] ⊂ monotone[PS, PS]**.

```
In[11]:= Map[subclass[P[enum[x]], #] &,
           SubstTest[intersection, monotone[restrict[x, u, u], restrict[y, v, v]],
            P[cart[u, v]], {u → OMEGA, v → OMEGA, x → PS, y → PS}]]

Out[11]= subclass[composite[enum[x], PS, inverse[enum[x]]], S] == True

In[12]:= subclass[composite[enum[x_], PS, inverse[enum[x_]]], S] := True
```

Comment. In the preceding theorem, one may notice that the second occurrence of the proper subset relation **PS** was automatically replaced with the subset relation **S**. This harmless rewriting is a consequence of the following rewrite rule.

```
In[13]:= subclass[x, PS]

Out[13]= and[equal[0, fix[x]], subclass[x, S]]
```

Corollary. A weak form of monotonicity: **P[enum[x]] ⊂ monotone[S, S]**.

```
In[14]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w], {u → P[enum[x]],
           v -> intersection[FUNS, monotone[PS, PS]], w -> monotone[S, S]}] // Reverse

Out[14]= subclass[composite[enum[x], S, inverse[enum[x]]], S] == True
```

```
In[15]:= subclass[composite[enum[x_], S, inverse[enum[x_]]], S] := True
```

the inclusion enum[x] ⊂ S

It will be shown in this section that the ordinal listed at any point **y ∈ domain[enum[x]]** is at least as big as **y**. To derive this fact, the well ordering principle will be applied to show that the class **fix[E ∘ enum[x]]** is empty.

Temporary Lemma. The condition that **enum[x]** and **inverse[E]** are disjoint is equivalent to the inclusion **enum[x] ⊂ S**.

```
In[16]:= SubstTest[subclass, enum[x], intersection[u, v],
          {u → cartsq[OMEGA], v → complement[inverse[E]]}]
Out[16]= equal[0, fix[composite[enum[x], E]]] == subclass[enum[x], S]

In[17]:= equal[0, fix[composite[enum[x_], E]]] := subclass[enum[x], S]
```

The condition **fix[u ∘ v] = 0** is equivalent to **fix[v ∘ u] = 0**. Because the function **enum[x]** is one-to-one, the composite **E ∘ enum[x]** is automatically rewritten as follows:

```
In[18]:= composite[E, enum[x]]
Out[18]= composite[inverse[IMAGE[inverse[enum[x]]], E]
```

Various lemmas will be derived to cope with this rewriting of **E ∘ enum[x]**.

Temporary Lemma. An inclusion.

```
In[19]:= SubstTest[subclass, fix[composite[E, w]], domain[w], w → enum[x]] // InvertFix // Reverse
Out[19]= subclass[fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]], domain[enum[x]]] == True

In[20]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary. A simplification rule.

```
In[21]:= equal[intersection[domain[enum[x]],
            fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]]],
           fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]]]
Out[21]= True

In[22]:= intersection[domain[enum[x_]],
            fix[composite[inverse[E], IMAGE[inverse[enum[x_]]]]]] :=
           fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]]
```

Corollary. Another condition equivalent to **enum[x] ⊂ S**.

```
In[23]:= SubstTest[empty, fix[composite[complement[t], funpart[w]]],
           {t → complement[E], w → enum[x]}] // Reverse // InvertFix

Out[23]= equal[0, fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]]] == subclass[enum[x], S]

In[24]:= equal[0, fix[composite[inverse[E], IMAGE[inverse[enum[x_]]]]]] := subclass[enum[x], S]
```

Membership statements of the form **t ∈ fix[E ∘ w]** are automatically rewritten as follows.

```
In[25]:= member[t, fix[composite[E, w]]]

Out[25]= member[t, image[inverse[w], t]]
```

Since **enum[x]** is a function this rewriting can be combined with the inverse image rule to yield the following.

Temporary Lemma. A rewrite rule for membership statements of the form **t ∈ fix[E ∘ enum[x]]**.

```
In[26]:= SubstTest[member, t, fix[composite[E, w]], w → enum[x]] // InvertFix // Reverse

Out[26]= member[t, fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]]] ==
          member[APPLY[enum[x], t], t]

In[27]:= member[t_, fix[composite[inverse[E], IMAGE[inverse[enum[x_]]]]]] :=
          member[APPLY[enum[x], t], t]
```

Temporary Lemma. The class **fix[E ∘ enum[x]]** is a subclass of **Ω**.

```
In[28]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]],
           subclass[u, w], {u -> fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]],
            v → domain[enum[x]], w → OMEGA}] // Reverse

Out[28]= subclass[fix[composite[inverse[E], IMAGE[inverse[enum[x]]]], OMEGA] == True

In[29]:= subclass[fix[composite[inverse[E], IMAGE[inverse[enum[x_]]]], OMEGA] := True
```

The main idea now is to show that this subclass of **Ω** is empty by applying the well-ordering principle.

Theorem. If **fix[E ∘ enum[x]]** is not empty, then it has a least member.

```
In[30]:= SubstTest[implies, subclass[t, OMEGA], or[empty[t], member[A[t], t]],
           t → fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]] // Reverse // MapNotNot

Out[30]= or[member[APPLY[enum[x], A[fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]]], A[
             fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]], subclass[enum[x], S]] == True

In[31]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. Introducing variables into a strict monotonicity statement.

```
In[32]:= SubstTest[implies, and[FUNCTION[w], subclass[P[w], monotone[x, y]],
           member[u, domain[w]], member[v, domain[w]], member[pair[u, v], x]],
         member[pair[APPLY[w, u], APPLY[w, v]], y], {w → enum[x], x → E, y → E}] // Reverse

Out[32]= or[member[APPLY[enum[x], u], APPLY[enum[x], v]], not[member[u, v]],
           not[member[u, domain[enum[x]]]], not[member[v, domain[enum[x]]]]] == True

In[33]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

One can do better because the domain of **enum[x]** is full.

Lemma.

```
In[34]:= SubstTest[implies, and[member[u, v], member[v, w], full[w]],
           member[u, w], w → domain[enum[x]]] // Reverse

Out[34]= or[member[u, domain[enum[x]]],
           not[member[u, v]], not[member[v, domain[enum[x]]]]] == True

In[35]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Theorem. Strict monotonicity of **enum[x]**.

```
In[36]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
            implies[and[p1, p2, p3], p4], not[implies[and[p1, p2], p4]],
            {p1 → member[u, v], p2 → member[v, domain[enum[x]]], p3 -> member[u, domain[enum[x]]],
             p4 -> member[APPLY[enum[x], u], APPLY[enum[x], v]]}]] // Reverse

Out[36]= or[member[APPLY[enum[x], u], APPLY[enum[x], v]],
           not[member[u, v]], not[member[v, domain[enum[x]]]]] == True

In[37]:= or[member[APPLY[enum[x_], u_], APPLY[enum[x_], v_]],
           not[member[u_, v_]], not[member[v_, domain[enum[x_]]]]] := True
```

Lemma. No element of **fix[E ∘ enum[x]]** is less than its least member (if there is one).

```
In[38]:= SubstTest[implies, subclass[t, OMEGA], disjoint[t, A[t]],
           t -> fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]]] // Reverse

Out[38]= equal[0, intersection[A[fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]]],
            fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]]] == True

In[39]:= intersection[A[fix[composite[inverse[E], IMAGE[inverse[enum[x_]]]]]],
           fix[composite[inverse[E], IMAGE[inverse[enum[x_]]]]] := 0
```

Corollary.

```
In[40]:= Map[not, SubstTest[member, t, intersection[u, v],
           {u -> A[fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]]],
            v -> fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]]}]]

Out[40]= or[not[member[t, A[fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]]]]],
           not[member[APPLY[enum[x], t], t]]] == True
```

```
In[41]:= or[not[member[t_, A[fix[composite[inverse[E], IMAGE[inverse[enum[x_]]]]]]]],
           not[member[APPLY[enum[x_], t_], t_]]] := True
```

Main Theorem. The function **enum[x]** is a subclass of the subset relation **S**.

```
In[42]:= (Map[not, SubstTest[and, implies[and[p0, p1], p2], implies[p2, p3], implies[p2, p4],
              implies[and[p3, p4], p5], implies[and[p4, p5], p6], implies[and[p0, p4], not[p6]],
              p0, p1, {p0 → equal[t, A[fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]]]],
               p1 → not[subclass[enum[x], S]], p2 → member[APPLY[enum[x], t], t],
               p3 → member[t, domain[enum[x]]], p4 → member[APPLY[enum[x], t], t],
               p5 → member[APPLY[enum[x], APPLY[enum[x], t]], APPLY[enum[x], t]],
               p6 → member[APPLY[enum[x], APPLY[enum[x], t]], APPLY[enum[x], t]]}]] // Reverse) /.
          t -> A[fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]]]

Out[42]= subclass[enum[x], S] == True
```

```
In[43]:= subclass[enum[x_], S] := True
```

Corollary. The ordinal listed by **enum[x]** at any point **y** is greater than or equal to **y**.

```
In[44]:= SubstTest[implies, and[member[r, s], subclass[s, t]], member[r, t],
           {r → pair[y, APPLY[enum[x], y]], s → enum[x], t → S}] // Reverse

Out[44]= or[not[member[y, domain[enum[x]]]], subclass[y, APPLY[enum[x], y]]] == True
```

```
In[45]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

A redundant literal can be removed here.

```
In[46]:= SubstTest[and, or[p, q], implies[p, q],
           {p -> member[y, domain[enum[x]]], q -> subclass[y, APPLY[enum[x], y]]}]

Out[46]= subclass[y, APPLY[enum[x], y]] == True
```

```
In[47]:= subclass[y_, APPLY[enum[x_], y_]] := True
```

Since both arguments and values of **enum[x]** are ordinals, one may prefer to replace **subclass** with **member**.

Lemma. Simplification rule.

```
In[48]:= equal[fix[composite[inverse[E], IMAGE[inverse[enum[x]]]]], 0]

Out[48]= True
```

```
In[49]:= fix[composite[inverse[E], IMAGE[inverse[enum[x_]]]]] := 0
```

Theorem. The ordinal listed by **enum[x]** at any point **y** cannot be less than **y**.

```
In[50]:= SubstTest[member, y, fix[composite[E, w]], w → enum[x]] // InvertFix

Out[50]= member[APPLY[enum[x], y], y] == False
```

```
In[51]:= member[APPLY[enum[x_], y_], y_] := False
```

corollaries

Several temporary rewrite rules encountered in the course of deriving **enum[x] ⊂ S** can be further simplified now that this inclusion is known to be true.

Theorem. The function **enum[x]** and the inverse membership relation are disjoint.

```
In[52]:= equal[intersection[inverse[E], enum[x]], 0]
Out[52]= True
In[53]:= intersection[enum[x_], inverse[E]] := 0
```

Corollary. A simplification rule.

```
In[54]:= equal[fix[composite[enum[x], E]], 0]
Out[54]= True
In[55]:= fix[composite[enum[x_], E]] := 0
```