enumerating all ordinals in a given class: part 2. properties of partial enumerations

Johan G. F. Belinfante
2010 October 5

```
In[1]:= SetDirectory["l:"]; << goedel.10oct04a

        :Package Title: goedel.10oct04a            2010 October 4 at 9:30 a.m.

        It is now:  2010 Oct 5 at 14:42

        Loading Simplification Rules

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

        weightlimit = 40
```

summary

This is the second notebook in a series about enumerating all ordinals in a given class in increasing order. Here the focus is mainly on properties of partial enumerations.

monotonicity of partial enumerations

Theorem. Partial enumerations are strictly monotone.

```
In[2]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p2, p3], not[implies[p1, p3]], {p1 → member[w, partenum[x]],
            p2 → subclass[w, composite[HULL[intersection[x, OMEGA]], IMAGE[w]]],
            p3 → subclass[w, composite[S, IMAGE[w]]]}]] // Reverse

Out[2]= or[not[member[w, partenum[x]]], subclass[w, composite[S, IMAGE[w]]]] == True

In[3]:= or[not[member[w_, partenum[x_]]], subclass[w_, composite[S, IMAGE[w_]]]] := True
```

Corollary. Every partial enumeration is strictly monotone.

```
In[4]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
           implies[and[p2, p3], p4], not[implies[p1, p4]], {p1 → member[w, partenum[x]],
            p2 → FUNCTION[w], p3 → subclass[w, composite[S, IMAGE[w]]],
            p4 → subclass[composite[w, E, inverse[w]], E]}]] // Reverse

Out[4]= or[not[member[w, partenum[x]]], subclass[composite[w, E, inverse[w]], E]] == True

In[5]:= or[not[member[w_, partenum[x_]]], subclass[composite[w_, E, inverse[w_]], E]] := True
```

Lemma. A temporary simplification rule introduced to aid in the elimination of variables.

```
In[6]:= implies[member[w, partenum[x]], member[w, monotone[E, E]]] // NotNotTest

Out[6]= or[and[subclass[w, cart[V, V]], subclass[composite[w, E, inverse[w]], E]],
          not[member[w, partenum[x]]]] == True

In[7]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Theorem. Eliminating the variable **w**.

```
In[8]:= Map[equal[V, #] &, SubstTest[class, w,
          implies[member[w, u], member[w, v]], {u → partenum[x], v → monotone[E, E]}]]

Out[8]= subclass[partenum[x], monotone[E, E]] == True

In[9]:= subclass[partenum[x_], monotone[E, E]] := True
```

ranges of partial enumerations

Lemma.

```
In[10]:= SubstTest[implies, subclass[x, t],
           subclass[range[x], range[t]], t -> composite[HULL[y], IMAGE[x]]] // Reverse

Out[10]= or[not[subclass[x, composite[HULL[y], IMAGE[x]]]],
           subclass[range[x], image[HULL[y], range[IMAGE[x]]]]] == True

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

Theorem. The range of any partial enumeration is a set of ordinals.

```
In[12]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
            implies[p3, p4], not[implies[p1, p4]], {p1 → member[w, partenum[x]],
             p2 → subclass[w, composite[HULL[intersection[OMEGA, x]], IMAGE[w]]],
             p3 → subclass[range[w], image[HULL[intersection[OMEGA, x]], range[IMAGE[w]]]],
             p4 → subclass[range[w], OMEGA]}]] // Reverse

Out[12]= or[not[member[w, partenum[x]]], subclass[range[w], OMEGA]] == True

In[13]:= or[not[member[w_, partenum[x_]]], subclass[range[w_], OMEGA]] := True
```

Eliminating the variable **w** yields a similar result for **enum[x]**.

Theorem. The range of **enum[x]** is a class of ordinals (not necessarily a set).

```
In[14]:= Map[equal[V, #] &, SubstTest[class, w, implies[member[w, u], member[w, v]],
           {u → partenum[x], v → image[inverse[IMAGE[SECOND]], P[OMEGA]]}]]

Out[14]= subclass[range[enum[x]], OMEGA] == True
```

```
In[15]:= subclass[range[enum[x_]], OMEGA] := True
```

Corollary. The class **enum[x]** is a subclass of the cartesian square of **Ω**.

```
In[16]:= SubstTest[subclass, composite[Id, t], cartsq[OMEGA], t → enum[x]] // Reverse

Out[16]= subclass[enum[x], cart[OMEGA, OMEGA]] == True
```

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

Any strictly monotone function whose domain and range are classes of ordinals is one-to-one.

Theorem. The class of partial enumerations is a class of bijections.

```
In[18]:= SubstTest[implies,
           and[subclass[u, v], subclass[v, w]], subclass[u, w], {u -> partenum[x],
            v -> intersection[FUNS, monotone[E, E], P[cartsq[OMEGA]]], w → BIJ}] // Reverse

Out[18]= subclass[partenum[x], BIJ] == True
```

```
In[19]:= subclass[partenum[x_], BIJ] := True
```

Corollary. Each partial enumeration is one-to-one.

```
In[20]:= SubstTest[implies, and[member[w, u], subclass[u, v]],
            member[w, v], {u -> partenum[x], v -> BIJ}] // MapNotNot // Reverse

Out[20]= or[FUNCTION[inverse[w]], not[member[w, partenum[x]]]] == True
```

```
In[21]:= or[FUNCTION[inverse[w_]], not[member[w_, partenum[x_]]]] := True
```

---

an alternate form of strict monotonicity

Lemma.

```
In[22]:= Map[subclass[#, monotone[complement[S], complement[S]]] &,
           SubstTest[intersection, P[cart[w, w]], monotone[restrict[t, w, w], restrict[t, w, w]],
            {t → complement[S], w → OMEGA}]] // Reverse

Out[22]= subclass[intersection[monotone[E, E], P[cart[OMEGA, OMEGA]]],
           monotone[complement[S], complement[S]]] == True
```

```
In[23]:= subclass[intersection[monotone[E, E], P[cart[OMEGA, OMEGA]]],
           monotone[complement[S], complement[S]]] := True
```

Theorem. An alternate formulation of strict monotonicity for partial enumerations.

```
In[24]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u → partenum[x], v -> intersection[monotone[E, E], P[cart[OMEGA, OMEGA]]],
            w -> monotone[complement[S], complement[S]]}] // Reverse

Out[24]= subclass[partenum[x], monotone[complement[S], complement[S]]] == True

In[25]:= subclass[partenum[x_], monotone[complement[S], complement[S]]] := True
```

Corollary.

```
In[27]:= SubstTest[implies, and[member[w, u], subclass[u, v]], member[w, v], {u → partenum[x],
             v -> monotone[complement[S], complement[S]]}] // Reverse // MapNotNot

Out[27]= or[not[member[w, partenum[x]]], subclass[composite[inverse[w], S, w], S]] == True

In[28]:= or[not[member[w_, partenum[x_]]], subclass[composite[inverse[w_], S, w_], S]] := True
```

---

recursion equations

Because any partial enumeration is a function, the recursion inclusion can be sharpened by introducing a new variable to an equation involving function application. The following general result will be used.

Theorem. The value of a function at any point of its domain is the same as that of any extension.

```
In[29]:= SubstTest[implies, equal[x, t], equal[APPLY[x, z], APPLY[t, z]],
           t → composite[funpart[y], id[domain[x]]]] // Reverse

Out[29]= or[equal[APPLY[x, z], APPLY[funpart[y], z]],
           not[member[z, domain[x]]], not[subclass[x, funpart[y]]]] == True

In[30]:= or[equal[APPLY[x_, z_], APPLY[funpart[y_], z_]],
           not[member[z_, domain[x_]]], not[subclass[x_, funpart[y_]]]] := True
```

The recursion condition $\mathbf{w \subset HULL[x] \circ IMAGE[w]}$ can be viewed as a statement that the function on the right side is an extension of $\mathbf{w}$.

Lemma. Restatement of the recursion inclusion as an equation.

```
In[31]:= SubstTest[implies, and[member[t, domain[u]], subclass[u, funpart[v]]],
           equal[APPLY[u, t], APPLY[funpart[v], t]], v → composite[HULL[x], IMAGE[u]]] // Reverse

Out[31]= or[equal[APPLY[u, t], hull[x, image[u, t]]], not[member[t, domain[u]]],
           not[subclass[u, composite[HULL[x], IMAGE[u]]]]] == True

In[32]:= (% /. {t → t_, x → x_, u → u_}) /. Equal → SetDelayed
```

Theorem. The ordinal listed in at any point in a partial enumeration is the least ordinal in x greater than all ordinals listed previously.

```
In[33]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[and[p0, p2], p3], not[implies[and[p0, p1], p3]],
           {p0 → member[t, domain[w]], p1 → member[w, partenum[x]],
            p2 → subclass[w, composite[HULL[intersection[OMEGA, x]], IMAGE[w]]],
            p3 -> equal[APPLY[w, t], hull[intersection[OMEGA, x], image[w, t]]]}]] // Reverse

Out[33]= or[equal[APPLY[w, t], hull[intersection[OMEGA, x], image[w, t]]],
           not[member[t, domain[w]]], not[member[w, partenum[x]]]] == True

In[34]:= or[equal[APPLY[w_, t_], hull[intersection[OMEGA, x_], image[w_, t_]]],
           not[member[t_, domain[w_]]], not[member[w_, partenum[x_]]]] := True
```

As a corollary, the following explicit formulation of the monotonicity condition for partial enumerations is obtained. Note that if **t** does not belong to the domain of **w**, then **APPLY[w, t] = V**, and the inclusion **image[w, t] ⊂ APPLY[w, t]** remains true.

Corollary. The ordinal listed at any point in a partial enumeration exceeds all previously listed ordinals.

```
In[35]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[p3, p4], or[p1, p4],
           not[implies[p2, p4]], {p1 → member[t, domain[w]], p2 -> member[w, partenum[x]],
            p3 -> equal[APPLY[w, t], hull[intersection[OMEGA, x], image[w, t]]],
            p4 -> subclass[image[w, t], APPLY[w, t]]}]] // Reverse

Out[35]= or[not[member[w, partenum[x]]], subclass[image[w, t], APPLY[w, t]]] == True

In[36]:= or[not[member[w_, partenum[x_]]], subclass[image[w_, t_], APPLY[w_, t_]]] := True
```

truncating partial enumerations

In this section it is shown that the restriction of any partial enumeration to any ordinal is again a partial enumeration.

Lemma. A simplification rule.

```
In[37]:= SubstTest[subclass, composite[u, id[ord[v]]],
          composite[t, id[ord[v]]], t → composite[x, IMAGE[id[ord[v]]]]]

Out[37]= subclass[composite[u, id[ord[v]]], composite[x, IMAGE[id[ord[v]]]]] ==
          subclass[composite[u, id[ord[v]]], x]

In[38]:= subclass[composite[u_, id[ord[v_]]], composite[x_, IMAGE[id[ord[v_]]]]] :=
          subclass[composite[u, id[ord[v]]], x]
```

Theorem. The restriction of any partial enumeration to any ordinal is a partial enumeration.

```
In[39]:= (Map[not, SubstTest[and, implies[and[p0, p2], p4],
            implies[and[p0, p3], p5], not[implies[and[p0, p1], p6]],
            {p0 -> equal[w, composite[u, id[ord[v]]]] , p1 → member[u, partenum[x]],
             p2 -> subclass[u, composite[HULL[intersection[OMEGA, x]], IMAGE[u]]],
             p3 → member[domain[u], OMEGA],
             p4 -> subclass[w, composite[HULL[intersection[OMEGA, x]], IMAGE[w]]],
             p5 → member[domain[w], OMEGA], p6 → member[w, partenum[x]]}]] /.
          w -> composite[u, id[ord[v]]]) // Reverse

Out[39]= or[member[composite[u, id[ord[v]]], partenum[x]], not[member[u, partenum[x]]]] == True

In[40]:= or[member[composite[u_, id[ord[v_]]], partenum[x_]],
           not[member[u_, partenum[x_]]]] := True
```

Corollary. Removing the **ord** wrapper.

```
In[41]:= SubstTest[implies, equal[v, ord[t]], or[member[composite[u, id[v]], partenum[x]],
            not[member[u, partenum[x]]]], t → v] // Reverse

Out[41]= or[member[composite[u, id[v]], partenum[x]],
           not[member[u, partenum[x]]], not[member[v, OMEGA]]] == True

In[42]:= or[member[composite[u_, id[v_]], partenum[x_]],
           not[member[u_, partenum[x_]]], not[member[v_, OMEGA]]] := True
```

---

domains of partial enumerations

The result of the preceding section is used to show that the class **image[IMAGE[FIRST], partenum[x]]** of domains of partial enumerations is a full subclass of **Ω**.

Lemma.

```
In[43]:= SubstTest[implies, member[t, z],
           member[domain[t], image[IMAGE[FIRST], z]], t → composite[x, id[y]]] // Reverse

Out[43]= or[member[intersection[y, domain[x]], image[IMAGE[FIRST], z]],
           not[member[composite[x, id[y]], z]]] == True

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

Theorem.

```
In[45]:= SubstTest[implies, equal[t, intersection[y, domain[x]]],
           or[member[t, image[IMAGE[FIRST], z]], not[member[composite[x, id[y]], z]]],
           t → y] // Reverse

Out[45]= or[member[y, image[IMAGE[FIRST], z]],
           not[member[composite[x, id[y]], z]], not[subclass[y, domain[x]]]] == True

In[46]:= or[member[y_, image[IMAGE[FIRST], z_]],
           not[member[composite[x_, id[y_]], z_]], not[subclass[y_, domain[x_]]]] := True
```

In the next lemma two steps are omitted to speed things up: **implies[and[p2, p3], p4]** and **implies[and[p5, p6], p7]**.

Lemma. Every ordinal less than the domain of some partial enumeration is also the domain of some partial enumeration.

```
In[47]:= Map[not,
           SubstTest[and, implies[p1, p3], implies[and[p1, p4], p5], implies[and[p2, p3], p6],
            not[implies[and[p1, p2], p7]], {p1 → member[u, partenum[x]],
             p2 → member[v, domain[u]], p3 → member[domain[u], OMEGA], p4 → member[v, OMEGA],
             p5 → member[composite[u, id[v]], partenum[x]], p6 → subclass[v, domain[u]],
             p7 → member[v, image[IMAGE[FIRST], partenum[x]]]}]] // Reverse

Out[47]= or[member[v, image[IMAGE[FIRST], partenum[x]]],
           not[member[u, partenum[x]]], not[member[v, domain[u]]]] == True

In[48]:= (% /. {u → u_, v → v_, x → x_}) /. Equal→ SetDelayed
```

Eliminating the variables **u** and **v** yields the following statement.

Lemma.

```
In[49]:= Map[empty[composite[Id, complement[#]]] &, SubstTest[class, pair[u, v],
           or[member[v, s], not[member[u, partenum[x]]], not[member[v, domain[u]]]],
           {s -> image[IMAGE[FIRST], partenum[x]], t → partenum[x]}]]

Out[49]= subclass[domain[enum[x]], image[IMAGE[FIRST], partenum[x]]] == True

In[50]:= subclass[domain[enum[x_]], image[IMAGE[FIRST], partenum[x_]]] := True
```

Restatement: The class of domains of partial enumerations is full.

```
In[51]:= full[image[IMAGE[FIRST], partenum[x]]]

Out[51]= True
```

Any full subclass of **Ω** is either an ordinal or the class **Ω** of all ordinals.

Corollary. The class of domains of partial enumerations for any class is either an ordinal or the class of all ordinals.

```
In[52]:= SubstTest[and, full[t], subclass[t, OMEGA], t → image[IMAGE[FIRST], partenum[x]]]

Out[52]= or[equal[OMEGA, image[IMAGE[FIRST], partenum[x]]],
           member[image[IMAGE[FIRST], partenum[x]], OMEGA]] == True

In[53]:= or[equal[OMEGA, image[IMAGE[FIRST], partenum[x_]]],
           member[image[IMAGE[FIRST], partenum[x_]], OMEGA]] := True
```