the class ENUMS

Johan G. F. Belinfante
2010 December 3

```
In[1]:= SetDirectory["l:"]; << goedel.10dec01a

        :Package Title: goedel.10dec01a            2010 December 1 at 12:30 noon

        It is now:  2010 Dec 3 at 8:19

        Loading Simplification Rules

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

        weightlimit = 40
```

summary

The definition of **enum[x]** as the union of partial enumerations of the ordinals in the class **x** does not provide a membership rule, and makes it difficult to quantify over the variable **x**. It is nonetheless possible to derive variable-free statements about all enumerations by exploiting the fact that **enum[x]** is the unique function contained in **Ω × Ω** that subcommutes with the membership relation and whose range is the class of ordinals in **x**. In this notebook the class **ENUMS** of all (small) enumerations is defined, and some of its properties are derived.

introduction

In this section a rewrite rule is derived that allows one to regard **enum[x]** as a wrapper for enumerations.

Lemma.

```
In[4]:= SubstTest[subcommute, funpart[t], E, t → enum[x]] // Reverse

Out[4]= subclass[composite[enum[x], E], composite[inverse[IMAGE[inverse[enum[x]]]], E]] == True

In[5]:= subclass[composite[enum[x_], E],
          composite[inverse[IMAGE[inverse[enum[x_]]]], E]] := True
```

The following rewrite rule characterizing enumerations resembles a wrapper-removal rule.

Theorem.

```
In[6]:= equiv[equal[x, enum[range[x]]], and[FUNCTION[x],
          subclass[x, cart[OMEGA, OMEGA]], subclass[composite[x, E], composite[E, x]]]]

Out[6]= True
```

```
In[7]:= equal[x_, enum[range[x_]]] := and[FUNCTION[x],
          subclass[x, cart[OMEGA, OMEGA]], subclass[composite[x, E], composite[E, x]]]
```

definition

The class **ENUMS** of all (small) enumerations is defined by the following equation.

```
In[8]:= intersection[FUNS,
          image[INVERSE, subcommutant[inverse[E]]], P[cart[OMEGA, OMEGA]]] := ENUMS
```

It is easier to reason about the class **ENUMS** if membership statements do not automatically expand out. Accordingly, only theorems about membership will be derived.

Theorem.

```
In[9]:= Map[implies[#, FUNCTION[x]] &, SubstTest[member, x, intersection[u, v],
           {u -> intersection[FUNS, image[INVERSE, subcommutant[inverse[E]]]],
            v → P[cartsq[OMEGA]]}]] // Reverse

Out[9]= or[FUNCTION[x], not[member[x, ENUMS]]] == True

In[10]:= or[FUNCTION[x_], not[member[x_, ENUMS]]] := True
```

Theorem.

```
In[11]:= Map[implies[#, subcommute[x, E]] &, SubstTest[member, x, intersection[u, v],
            {u -> intersection[FUNS, image[INVERSE, subcommutant[inverse[E]]]],
             v → P[cartsq[OMEGA]]}]] // Reverse

Out[11]= or[not[member[x, ENUMS]], subclass[composite[x, E], composite[E, x]]] == True

In[12]:= or[not[member[x_, ENUMS]], subclass[composite[x_, E], composite[E, x_]]] := True
```

Theorem.

```
In[13]:= Map[implies[#, subclass[x, cartsq[OMEGA]]] &, SubstTest[member, x, intersection[u, v],
            {u -> intersection[FUNS, image[INVERSE, subcommutant[inverse[E]]]],
             v → P[cartsq[OMEGA]]}]] // Reverse

Out[13]= or[not[member[x, ENUMS]], subclass[x, cart[OMEGA, OMEGA]]] == True

In[14]:= or[not[member[x_, ENUMS]], subclass[x_, cart[OMEGA, OMEGA]]] := True
```

In the reverse direction, one has:

Theorem.

```
In[15]:= Map[
           implies[and[member[x, y], FUNCTION[x], subcommute[x, E], subclass[x, cartsq[OMEGA]]],
             #] &, SubstTest[member, x, intersection[u, v],
            {u -> intersection[FUNS, image[INVERSE, subcommutant[inverse[E]]]],
             v → P[cartsq[OMEGA]]}]] // Reverse

Out[15]= or[member[x, ENUMS], not[FUNCTION[x]],
           not[member[x, y]], not[subclass[x, cart[OMEGA, OMEGA]]],
           not[subclass[composite[x, E], composite[E, x]]]] == True

In[16]:= or[member[x_, ENUMS], not[FUNCTION[x_]],
           not[member[x_, y_]], not[subclass[x_, cart[OMEGA, OMEGA]]],
           not[subclass[composite[x_, E], composite[E, x_]]]] := True
```

The following may nonetheless be convenient.

Theorem.

```
In[29]:= equiv[and[FUNCTION[x], member[x, V], subclass[x, cart[OMEGA, OMEGA]],
             subclass[composite[x, E], composite[E, x]]], member[x, ENUMS]] // not // not

Out[29]= True

In[30]:= and[FUNCTION[x_], member[x_, V], subclass[x_, cart[OMEGA, OMEGA]],
           subclass[composite[x_, E], composite[E, x_]]] := member[x, ENUMS]
```

enum[x] ∈ ENUMS

Lemma.

```
In[21]:= SubstTest[subclass, inverse[u], inverse[v],
          {u -> composite[inverse[E], inverse[enum[x]]],
           v -> composite[inverse[E], IMAGE[inverse[enum[x]]]]}]

Out[21]= subclass[composite[inverse[E], inverse[enum[x]]],
           composite[inverse[E], IMAGE[inverse[enum[x]]]]] == True

In[22]:= subclass[composite[inverse[E], inverse[enum[x_]]],
           composite[inverse[E], IMAGE[inverse[enum[x_]]]]] := True
```

Theorem.

```
In[23]:= SubstTest[member, enum[x], intersection[u, v, w], {u → FUNS,
            v -> image[INVERSE, subcommutant[inverse[E]]], w → P[cartsq[OMEGA]]}] // Reverse

Out[23]= member[enum[x], ENUMS] == member[intersection[OMEGA, x], V]

In[24]:= member[enum[x_], ENUMS] := member[intersection[OMEGA, x], V]
```

With this rule in place, the wrapper removal rule yields:

```
In[31]:= and[equal[x, enum[range[x]]], member[x, V]]

Out[31]= member[x, ENUMS]
```

ranges of enumerations

Lemma.

```
In[32]:= SubstTest[implies, member[pair[x, y], composite[Id, t]], member[x, domain[t]],
           {t -> composite[id[ENUMS], inverse[IMAGE[SECOND]]], y → enum[x]}] // Reverse

Out[32]= or[member[x, image[IMAGE[SECOND], ENUMS]],
           not[member[x, V]], not[subclass[x, OMEGA]]] == True

In[33]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. (Eliminate the variable **x**.)

```
In[34]:= Map[equal[V, #] &, SubstTest[class, x, implies[member[x, u], member[x, v]],
           {u → P[OMEGA], v -> image[IMAGE[SECOND], ENUMS]}]]

Out[34]= subclass[P[OMEGA], image[IMAGE[SECOND], ENUMS]] == True

In[35]:= subclass[P[OMEGA], image[IMAGE[SECOND], ENUMS]] := True
```

Lemma.

```
In[40]:= SubstTest[implies, and[member[x, v], equal[x, enum[range[x]]]],
           subclass[range[x], OMEGA], v → V] // Reverse

Out[40]= or[not[member[x, ENUMS]], subclass[range[x], OMEGA]] == True

In[41]:= or[not[member[x_, ENUMS]], subclass[range[x_], OMEGA]] := True
```

Lemma. (Eliminating the variable **x**.)

```
In[42]:= Map[equal[V, #] &, SubstTest[class, x, implies[member[x, u], member[x, v]],
           {u → ENUMS, v → image[inverse[IMAGE[SECOND]], P[OMEGA]]}]]

Out[42]= subclass[range[U[ENUMS]], OMEGA] == True

In[43]:= % /. Equal → SetDelayed
```

Theorem.

```
In[44]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> P[OMEGA], v -> image[IMAGE[SECOND], ENUMS]}]

Out[44]= equal[image[IMAGE[SECOND], ENUMS], P[OMEGA]] == True

In[45]:= image[IMAGE[SECOND], ENUMS] := P[OMEGA]
```

domains of enumerations

Theorem.

```
In[55]:= SubstTest[implies, and[member[x, V], equal[x, enum[t]]],
           member[domain[x], OMEGA], t → range[x]] // Reverse

Out[55]= or[member[domain[x], OMEGA], not[member[x, ENUMS]]] == True

In[56]:= or[member[domain[x_], OMEGA], not[member[x_, ENUMS]]] := True
```

Lemma. (Eliminate the variable **x**.)

```
In[58]:= Map[equal[V, #] &, SubstTest[class, x, implies[member[x, u], member[x, v]],
           {u → ENUMS, v → image[inverse[IMAGE[FIRST]], OMEGA]}]]

Out[58]= subclass[image[IMAGE[FIRST], ENUMS], OMEGA] == True

In[59]:= % /. Equal → SetDelayed
```

An inclusion in the opposite direction can be derived by considering enumerations of ordinals.

Lemma. The identity function on any ordinal is an enumeration.

```
In[62]:= SubstTest[member, enum[t], ENUMS, t → ord[x]] // Reverse

Out[62]= member[id[ord[x]], ENUMS] == True

In[63]:= member[id[ord[x_]], ENUMS] := True
```

Corollary. A simple special case.

```
In[68]:= SubstTest[member, id[ord[x]], ENUMS, x → 0] // Reverse

Out[68]= member[0, ENUMS] == True

In[69]:= member[0, ENUMS] := True
```

Theorem.

```
In[70]:= Map[empty,
          SubstTest[reify, x, dif[set[ord[x]], t], t → image[inverse[IMAGE[DUP]], ENUMS]]]

Out[70]= subclass[image[IMAGE[DUP], OMEGA], ENUMS] == True

In[71]:= subclass[image[IMAGE[DUP], OMEGA], ENUMS] := True
```

Corollary. An inclusion in the opposite direction form an earlier one.

```
In[72]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
           {t → IMAGE[FIRST], u -> image[IMAGE[DUP], OMEGA], v -> ENUMS}] // Reverse

Out[72]= subclass[OMEGA, image[IMAGE[FIRST], ENUMS]] == True

In[73]:= % /. Equal → SetDelayed
```

Theorem. The class of domains of (small) enumerations is **Ω**.

```
In[74]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> image[IMAGE[FIRST], ENUMS], v → OMEGA}]

Out[74]= equal[OMEGA, image[IMAGE[FIRST], ENUMS]] == True

In[75]:= image[IMAGE[FIRST], ENUMS] := OMEGA
```

a one-to-one correspondence

Lemma.

```
In[76]:= SubstTest[implies, equal[u, v], equal[enum[u], enum[v]],
           {u → intersection[OMEGA, x], v → intersection[OMEGA, y]}] // Reverse

Out[76]= or[equal[enum[x], enum[y]],
           not[equal[intersection[OMEGA, x], intersection[OMEGA, y]]]] == True

In[77]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. Enumerations with the same range are equal.

```
In[80]:= SubstTest[implies, and[equal[x, enum[u]], equal[y, enum[v]]],
            implies[member[pair[x, y], composite[inverse[IMAGE[SECOND]], IMAGE[SECOND]]],
             equal[x, y]], {u → range[x], v → range[y]}] // MapNotNot // Reverse

Out[80]= or[equal[x, y], not[equal[range[x], range[y]]],
           not[member[x, ENUMS]], not[member[y, ENUMS]]] == True

In[81]:= or[equal[x_, y_], not[equal[range[x_], range[y_]]],
           not[member[x_, ENUMS]], not[member[y_, ENUMS]]] := True
```

Eliminating the variables yields the following.

Theorem. The function **IMAGE[SECOND]** restricted to **ENUMS** is one-to-one.

```
In[82]:= Map[empty[domain[complement[#]]] &, SubstTest[class, pair[u, v],
           implies[member[pair[u, v], composite[t, inverse[t]]], equal[u, v]],
           t -> composite[id[ENUMS], inverse[IMAGE[SECOND]]]]]

Out[82]= FUNCTION[composite[id[ENUMS], inverse[IMAGE[SECOND]]]] == True

In[83]:= FUNCTION[composite[id[ENUMS], inverse[IMAGE[SECOND]]]] := True
```

This provides a one-to-one correspondence between sets of ordinals and enumerations. The composite of this function with **IMAGE[FIRST]** yields a one-to-one correspondence betweem sets of ordinals and ordinals.

Corollary.

```
In[87]:= SubstTest[FUNCTION, composite[funpart[u], funpart[v]],
           {u -> IMAGE[FIRST], v -> composite[id[ENUMS], inverse[IMAGE[SECOND]]]}] // Reverse

Out[87]= FUNCTION[inverse[image[DORA, ENUMS]]] == True

In[88]:= FUNCTION[inverse[image[DORA, ENUMS]]] := True
```

Comment. The function **inverse[image[DORA, ENUMS]]** assigns an ordinal to each set of ordinals, usually called its **order type**.

closure under unions of chains

In this section it is shown that the class of enumerations is closed under unions of chains. This is true because each of the three classes whose intersection is **ENUMS** also has that property. In fact, two of these three classes is closed under arbitrary unions. The class **FUNS** is the only on of the three that is not closed under arbitrary unions.

Lemma. (Closure under arbitrary unions implies closure under unions of chains.)

```
In[91]:= SubstTest[Uchains, Uclosure[t], t → image[INVERSE, subcommutant[x]]] // Reverse

Out[91]= Uchains[image[INVERSE, subcommutant[x]]] == image[INVERSE, subcommutant[x]]

In[92]:= Uchains[image[INVERSE, subcommutant[x_]]] := image[INVERSE, subcommutant[x]]
```

Theorem. The class **ENUMS** is closed under unions of chains.

```
In[93]:= SubstTest[implies, and[equal[x, Uchains[x]], equal[y, Uchains[y]]],
           equal[intersection[x, y], Uchains[intersection[x, y]]]], {x → FUNS, y -> intersection[
              image[INVERSE, subcommutant[inverse[E]]], P[cart[OMEGA, OMEGA]]]}] // Reverse

Out[93]= equal[ENUMS, Uchains[ENUMS]] == True

In[94]:= Uchains[ENUMS] := ENUMS
```

enumerations are one-to-one

Lemma.

```
In[96]:= SubstTest[implies, and[member[x, V], equal[x, enum[t]]],
            member[x, BIJ], t → range[x]] // MapNotNot // Reverse

Out[96]= or[FUNCTION[inverse[x]], not[member[x, ENUMS]]] == True
```

In[97]:= **or[FUNCTION[inverse[x_]], not[member[x_, ENUMS]]] := True**

Lemma.

In[103]:=
```
member[x, dif[ENUMS, BIJ]] // NotNotTest
```

Out[103]=
```
or[and[member[x, ENUMS], not[FUNCTION[x]]],
  and[member[x, ENUMS], not[FUNCTION[inverse[x]]]]] == False
```

In[104]:=
```
(% /. x → x_) /. Equal → SetDelayed
```

Theorem. Every enumeration is a bijection.

In[105]:=
```
Map[empty, dif[ENUMS, BIJ] // Normality]
```

Out[105]=
```
subclass[ENUMS, BIJ] == True
```

In[106]:=
```
subclass[ENUMS, BIJ] := True
```

Corollary. Any set of ordinals is equipollent to its order type.

In[108]:=
```
SubstTest[implies, subclass[u, v],
  subclass[image[t, u], image[t, v]], {t → DORA, u → ENUMS, v → BIJ}] // Reverse
```

Out[108]=
```
subclass[image[DORA, ENUMS], Q] == True
```

In[109]:=
```
subclass[image[DORA, ENUMS], Q] := True
```