increasing maps are enumerations

Johan G. F. Belinfante
2010 October 30

```
In[1]:= SetDirectory["l:"]; << goedel.10oct29a

        :Package Title: goedel.10oct29a            2010 October 29 at 2:50 p.m.

        It is now:  2010 Oct 30 at 9:31

        Loading Simplification Rules

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

        weightlimit = 40
```

summary

In this notebook a characterization of the function **enum[x]** which lists all the ordinals in a class **x** in increasing order is derived. It is shown that any function **x ⊂ Ω × Ω** that subcommutes with the membership relation **E** is the enumeration of its range: **x = enum[range[x]]**.

monotonicity results

Theorem. The inverse of a function **x ⊂ Ω × Ω** that subcommutes with the membership relation **E** is monotone with respect to inclusion, **P[inverse[x]] ⊂ monotone[S, S]**.

```
In[2]:= Map[not, SubstTest[and, implies[and[p1, p3], p4], implies[and[p2, p4], p5],
           not[implies[and[p1, p2, p3], p5]], {p1 → FUNCTION[x], p2 → subclass[x, cartsq[OMEGA]],
            p3 → subcommute[x, E], p4 → subclass[composite[x, E, inverse[x]], E],
            p5 -> subclass[composite[inverse[x], S, x], S]}]] // Reverse

Out[2]= or[not[FUNCTION[x]], not[subclass[x, cart[OMEGA, OMEGA]]],
          not[subclass[composite[x, E], composite[E, x]]],
          subclass[composite[inverse[x], S, x], S]] == True

In[3]:= or[not[FUNCTION[x_]], not[subclass[x_, cart[OMEGA, OMEGA]]],
          not[subclass[composite[x_, E], composite[E, x_]]],
          subclass[composite[inverse[x_], S, x_], S]] := True
```

Theorem. If the inverse of a function **x ⊂ Ω × Ω** is monotone with respect to inclusion, and if **domain[x]** is full, then **x** subcommutes with the membership relation.

```
In[4]:= Map[not, SubstTest[and, implies[and[p2, p4], p5],
           implies[and[p1, p3, p5], p6], not[implies[and[p1, p2, p3, p4], p6]],
           {p1 → FUNCTION[x], p2 -> subclass[x, cartsq[OMEGA]],
            p3 → full[domain[x]], p4 -> subclass[composite[inverse[x], S, x], S],
            p5 -> subclass[composite[x, E, inverse[x]], E], p6 → subcommute[x, E]}]] // Reverse

Out[4]= or[not[FUNCTION[x]], not[subclass[x, cart[OMEGA, OMEGA]]],
          not[subclass[composite[inverse[x], S, x], S]], not[subclass[U[domain[x]], domain[x]]],
          subclass[composite[x, E], composite[E, x]]] == True

In[5]:= or[not[FUNCTION[x_]], not[subclass[composite[inverse[x_], S, x_], S]],
          not[subclass[x_, cart[OMEGA, OMEGA]]], not[subclass[U[domain[x_]], domain[x_]]],
          subclass[composite[x_, E], composite[E, x_]]] := True
```

---

an abbreviation

The main strategy of the derivation to be presented below is to apply an ordinal rigidity theorem to the following class, for which a temporary abbreviation is introduced.

```
In[6]:= composite[inverse[enum[range[x_]]], x_] := tmp[x]
```

Some useful rewrite rules for **tmp[x]** follow immediately from this definition.

Lemma. A simplification rule for the inverse.

```
In[7]:= composite[inverse[x], enum[range[x]]] // DoubleInverse

Out[7]= composite[inverse[x], enum[range[x]]] == inverse[tmp[x]]

In[8]:= composite[inverse[x_], enum[range[x_]]] := inverse[tmp[x]]
```

Theorem. A simplication rule.

```
In[9]:= Assoc[enum[range[x]], inverse[enum[range[x]]], x]

Out[9]= composite[enum[range[x]], tmp[x]] == composite[id[OMEGA], x]

In[10]:= composite[enum[range[x_]], tmp[x_]] := composite[id[OMEGA], x]
```

Theorem. A simplification rule for the domain.

```
In[11]:= IminComp[inverse[enum[range[x]]], x, V] // Reverse

Out[11]= image[inverse[x], intersection[OMEGA, range[x]]] == domain[tmp[x]]

In[12]:= image[inverse[x_], intersection[OMEGA, range[x_]]] := domain[tmp[x]]
```

Theorem. A formula for the range.

```
In[13]:= ImageComp[inverse[enum[range[x]]], x, V]

Out[13]= range[tmp[x]] == domain[enum[range[x]]]

In[14]:= range[tmp[x_]] := domain[enum[range[x]]]
```

Lemma. A formula for the domain.

```
In[15]:= SubstTest[implies, subclass[range[x], t],
           equal[domain[x], image[inverse[x], t]], t → intersection[OMEGA, range[x]]] // Reverse

Out[15]= or[equal[domain[x], domain[tmp[x]]], not[subclass[range[x], OMEGA]]] == True

In[16]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. If **x** is a function, then so is **tmp[x]**.

```
In[17]:= SubstTest[implies, and[FUNCTION[t], FUNCTION[x]],
           FUNCTION[composite[t, x]], t → inverse[enum[range[x]]]] // Reverse

Out[17]= or[FUNCTION[tmp[x]], not[FUNCTION[x]]] == True

In[18]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. If $\mathbf{x} \subset \Omega \times \Omega$, then the same holds for **tmp[x]**.

```
In[19]:= (SubstTest[implies, and[subclass[t, w], subclass[x, w]], subclass[composite[t, x],
             composite[w, w]], t → inverse[enum[range[x]]]] /. w → cartsq[OMEGA]) // Reverse

Out[19]= or[not[subclass[x, cart[OMEGA, OMEGA]]], subclass[tmp[x], cart[OMEGA, OMEGA]]] == True

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

Theorem. If the inverse of **x** is monotone with respect to inclusion, then so is **tmp[x]**.

```
In[21]:= SubstTest[implies, and[subclass[composite[u, S, inverse[u]], S],
            subclass[composite[v, S, inverse[v]], S]],
           subclass[composite[u, v, S, inverse[v], inverse[u]], S],
           {u → inverse[x], v → enum[range[x]]}] // Reverse

Out[21]= or[not[subclass[composite[inverse[x], S, x], S]],
           subclass[composite[inverse[tmp[x]], S, tmp[x]], S]] == True

In[22]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. If $\mathbf{x} \subset \Omega \times \Omega$, then the domain of **tmp[x]** is equal to the domain of **x**.

```
In[23]:= Map[not, SubstTest[and, implies[p2, p4], implies[p4, p5],
            not[implies[p2, p5]], {p2 -> subclass[x, cartsq[OMEGA]],
             p4 → subclass[range[x], OMEGA], p5 → equal[domain[tmp[x]], domain[x]]}]] // Reverse

Out[23]= or[equal[domain[x], domain[tmp[x]]], not[subclass[x, cart[OMEGA, OMEGA]]]] == True

In[24]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. If $\mathbf{x \subset \Omega \times \Omega}$ subcommutes with **E**, then the domain of **tmp[x]** is full.

```
In[25]:= Map[not, SubstTest[and, implies[p2, p5],
            implies[and[p3, p5], p6], not[implies[and[p2, p3], p6]],
            {p1 → FUNCTION[x], p2 → subclass[x, cartsq[OMEGA]], p3 → subcommute[x, E],
             p5 → equal[domain[tmp[x]], domain[x]], p6 → full[domain[tmp[x]]]}]] // Reverse

Out[25]= or[not[subclass[x, cart[OMEGA, OMEGA]]],
           not[subclass[composite[x, E], composite[E, x]]],
           subclass[U[domain[tmp[x]]], domain[tmp[x]]]] == True

In[26]:= (% /. x → x_) /. Equal → SetDelayed
```

derivation of the main theorem

Theorem. Application of a rigidity theorem for strictly monotone functions whose domain and range are both full subclasses of $\mathbf{\Omega}$.

```
In[27]:= SubstTest[implies, and[FUNCTION[t], subclass[t, cartsq[OMEGA]],
            subclass[composite[inverse[t], S, t], S], full[domain[t]], full[range[t]]],
           subclass[t, Id], t -> tmp[x]] // Reverse

Out[27]= or[not[FUNCTION[tmp[x]]], not[subclass[composite[inverse[tmp[x]], S, tmp[x]], S]],
           not[subclass[tmp[x], cart[OMEGA, OMEGA]]],
           not[subclass[U[domain[tmp[x]]], domain[tmp[x]]], subclass[tmp[x], Id]] == True

In[28]:= (% /. x → x_) /. Equal → SetDelayed
```

To expedite execution, the proof step **implies[and[p3, p4, p5, p6], p7]** is omitted in the derivation of the next theorem.

Theorem. If a function $\mathbf{x \subset \Omega \times \Omega}$ subcommutes with **E**, then **tmp[x]** $\subset$ **Id**.

```
In[29]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
            implies[p2, p4], implies[p1, p5], implies[p1, p6], not[implies[p1, p7]],
            {p1 → and[FUNCTION[x], subclass[x, cartsq[OMEGA]], subcommute[x, E]],
             p2 → subclass[composite[inverse[x], S, x], S], p3 → FUNCTION[tmp[x]],
             p4 → subclass[composite[inverse[tmp[x]], S, tmp[x]], S],
             p5 → subclass[tmp[x], cartsq[OMEGA]], p6 -> full[domain[tmp[x]]],
             p7 → subclass[tmp[x], Id]}]] // Reverse

Out[29]= or[not[FUNCTION[x]], not[subclass[x, cart[OMEGA, OMEGA]]],
           not[subclass[composite[x, E], composite[E, x]]], subclass[tmp[x], Id]] == True

In[30]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. A consequence of **tmp[x]** $\subset$ **Id**.

```
In[31]:= SubstTest[implies, subclass[v, w], subclass[composite[u, v], composite[u, w]],
           {u → enum[range[x]], v -> composite[inverse[enum[range[x]]], x], w -> Id}] // Reverse

Out[31]= or[not[subclass[tmp[x], Id]], subclass[composite[id[OMEGA], x], enum[range[x]]]] == True

In[32]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. This lemma is needed to eliminate the factor **id[Ω]** in the preceding result.

```
In[33]:= (SubstTest[implies, and[equal[u, v], subclass[v, z]], subclass[u, z],
            {u → funpart[t], v -> composite[id[y], funpart[t]]}] // Reverse)

Out[33]= or[not[subclass[composite[id[y], funpart[t]], z]],
           not[subclass[range[funpart[t]], y]], subclass[funpart[t], z]] == True

In[34]:= (% /. {t → t_, y → y_, z → z_}) /. Equal → SetDelayed
```

Theorem. (Remove the **funpart** wrapper.)

```
In[35]:= SubstTest[implies, and[equal[x, funpart[t]], subclass[composite[id[y], x], z],
           subclass[range[x], y]], subclass[x, z], t → x] // Reverse

Out[35]= or[not[FUNCTION[x]], not[subclass[composite[id[y], x], z]],
           not[subclass[range[x], y]], subclass[x, z]] == True

In[36]:= or[not[FUNCTION[x_]], not[subclass[composite[id[y_], x_], z_]],
           not[subclass[range[x_], y_]], subclass[x_, z_]] := True
```

Lemma. The domain and range of an identity function are equal.

```
In[37]:= SubstTest[implies, subclass[t, Id], equal[domain[t], range[t]], t → tmp[x]] // Reverse

Out[37]= or[equal[domain[enum[range[x]]], domain[tmp[x]]], not[subclass[tmp[x], Id]]] == True

In[38]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. Equality of the domains of **x** and of **enum[range[x]]**.

```
In[39]:= Map[not, SubstTest[and, implies[p1, p3],
           implies[and[p2, p3], p4], not[implies[and[p1, p2], p4]],
           {p1 -> subclass[tmp[x], Id], p2 -> subclass[range[x], OMEGA], p3 ->
             equal[domain[enum[range[x]]], image[inverse[x], intersection[OMEGA, range[x]]]],
            p4 -> equal[domain[x], domain[enum[range[x]]]]}]] // Reverse

Out[39]= or[equal[domain[x], domain[enum[range[x]]]],
           not[subclass[range[x], OMEGA]], not[subclass[tmp[x], Id]]] == True

In[40]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. The only subclass of a function with the same domain is the function itself.

```
In[41]:= SubstTest[implies, and[FUNCTION[t], subclass[x, t], equal[domain[x], domain[t]]],
           equal[x, t], t → enum[y]] // Reverse

Out[41]= or[equal[x, enum[y]],
           not[equal[domain[x], domain[enum[y]]]], not[subclass[x, enum[y]]] == True

In[42]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

To speed up execution, these three steps are omitted in the derivation of the main theorem: **implies[p1, p4]**, **implies[p2, p3]** and **implies[and[p5, p6], p7]**.

Main Theorem. If a function $\mathbf{x \subset \Omega \times \Omega}$ subcommutes with **E**, then **x = enum[range[x]]**.

```
In[43]:= Map[not, SubstTest[and, implies[p1, p2],
            implies[and[p1, p3, p4], p5], implies[and[p2, p4], p6], not[implies[p1, p7]],
            {p1 -> and[FUNCTION[x], subclass[x, cartsq[OMEGA]], subcommute[x, E]],
             p2 → subclass[tmp[x], Id], p3 -> subclass[composite[id[OMEGA], x], enum[range[x]]],
             p4 → subclass[range[x], OMEGA], p5 → subclass[x, enum[range[x]]],
             p6 → equal[domain[x], domain[enum[range[x]]]],
             p7 → equal[x, enum[range[x]]]}]] // Reverse

Out[43]= or[equal[x, enum[range[x]]], not[FUNCTION[x]], not[subclass[x, cart[OMEGA, OMEGA]]],
           not[subclass[composite[x, E], composite[E, x]]]] == True

In[44]:= or[equal[x_, enum[range[x_]]],
           not[FUNCTION[x_]], not[subclass[x_, cart[OMEGA, OMEGA]]],
           not[subclass[composite[x_, E], composite[E, x_]]]] := True
```