reify rule for enum[x]

Johan G. F. Belinfante
2010 December 13

```
In[1]:= SetDirectory["l:"]; << goedel.10dec11a

        :Package Title: goedel.10dec11a              2010 December 11 at 6:45 p.m.

        It is now:  2010 Dec 13 at 18:25

        Loading Simplification Rules

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

        weightlimit = 40
```

summary

The image by **enum[x]** of any ordinal α belonging to its domain is $\mathbf{\Omega} \bigcap \mathbf{x} \bigcap$ **APPLY[enum[x],** α**]**. The class of all ranges of partial enumerations of the ordinals in a class **x** is the class of all intersections of **x** with ordinals. The class of all partial enumerations of the ordinals in a class **x** is the class of enumerations whose ranges are intersections of **x** with ordinals. These results are used to derive **reify** rules for **partenum** and **enum**.

a formula for image[enum[x], y]

At any point in the enumeration of ordinals in a class **x**, the set of ordinals already listed is the set of all ordinals less that the next one to be listed. This observation will be given a formal statement in this section.

Observation. An inclusion in one direction is already available:

```
In[2]:= subclass[image[enum[x], y], intersection[OMEGA, x, APPLY[enum[x], y]]]

Out[2]= True
```

To obtain an inclusion in the reverse direction, one has only to eliminate a variable.

Lemma.

```
In[3]:= Map[equal[V, #] &, SubstTest[class, u,
          implies[and[member[y, w], member[u, z]], member[u, y]], {w → domain[enum[x]],
           z -> image[inverse[enum[x]], intersection[OMEGA, x, APPLY[enum[x], y]]]}]]

Out[3]= or[not[member[y, domain[enum[x]]]], subclass[
           image[inverse[enum[x]], intersection[OMEGA, x, APPLY[enum[x], y]]], y]] == True
```

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

Combining this inclusion with the reverse inclusion yields an equation, the main result of this section.

Main Theorem. A formula for **image[enum[x], y]** for **y ∈ domain[enum[x]]**.

```
In[5]:= SubstTest[and, implies[p, subclass[u, v]],
          subclass[v, u], {p → member[y, domain[enum[x]]],
           u -> intersection[OMEGA, x, APPLY[enum[x], y]], v -> image[enum[x], y]}]

Out[5]= or[equal[image[enum[x], y], intersection[OMEGA, x, APPLY[enum[x], y]]],
          not[member[y, domain[enum[x]]]]] == True

In[6]:= or[equal[image[enum[x_], y_], intersection[OMEGA, x_, APPLY[enum[x_], y_]]],
          not[member[y_, domain[enum[x_]]]]] := True
```

In order to eliminate the variable **y** here it is convenient to restate this result as a simple equation by introducing an **image[V, -]** term for the condition **y ∈ domain[enum[x]]**.

Corollary. Restatement of the main result.

```
In[7]:= equal[intersection[OMEGA, x, APPLY[enum[x], y],
           image[V, intersection[domain[enum[x]], set[y]]]],
          intersection[image[V, intersection[domain[enum[x]], set[y]]], image[enum[x], y]]]

Out[7]= True

In[8]:= intersection[OMEGA, x_, APPLY[enum[x_], y_],
           image[V, intersection[domain[enum[x_]], set[y_]]]] :=
          intersection[image[V, intersection[domain[enum[x]], set[y]]], image[enum[x], y]]
```

The variable **y** can now be eliminated using **reify**.

Theorem.

```
In[9]:= Map[composite[VERTSECT[#], id[domain[enum[x]]]] &,
          SubstTest[reify, y, intersection[OMEGA, x, APPLY[t, y],
            image[V, intersection[domain[t], set[y]]]], t -> enum[x]]]

Out[9]= composite[IMAGE[id[intersection[OMEGA, x]]], enum[x]] ==
          composite[IMAGE[enum[x]], id[domain[enum[x]]]]

In[10]:= composite[IMAGE[id[intersection[OMEGA, x_]]], enum[x_]] :=
           composite[IMAGE[enum[x]], id[domain[enum[x]]]]
```

Corollary. A slightly simpler equation of the same type as the previous one.

```
In[11]:= Assoc[IMAGE[id[x]], IMAGE[id[OMEGA]], enum[x]]

Out[11]= composite[IMAGE[id[x]], enum[x]] == composite[IMAGE[enum[x]], id[domain[enum[x]]]]

In[12]:= composite[IMAGE[id[x_]], enum[x_]] := composite[IMAGE[enum[x]], id[domain[enum[x]]]]
```

a general result

In this section a corollary is derived that has does not mention enumerations, but follows from the results obtained above.

Lemma.

```
In[13]:= Map[equal[intersection[OMEGA, x], fix[#]] &,
          Assoc[composite[HULL[intersection[OMEGA, x]], IMAGE[id[intersection[OMEGA, x]]]],
           enum[x], inverse[enum[x]]]]

Out[13]= subclass[intersection[OMEGA, x],
           fix[composite[HULL[intersection[OMEGA, x]], IMAGE[id[x]]]]] == True

In[14]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[15]:= SubstTest[implies, equal[t, intersection[x, OMEGA]],
           subclass[t, fix[composite[HULL[t], IMAGE[id[x]]]]], t → x] // Reverse

Out[15]= or[not[subclass[x, OMEGA]], subclass[x, fix[composite[HULL[x], IMAGE[id[x]]]]]] == True

In[16]:= or[not[subclass[x_, OMEGA]],
           subclass[x_, fix[composite[HULL[x_], IMAGE[id[x_]]]]]] := True
```

ranges of partial enumerations

Returning to the theory of enumeration, in this section some groundwork is laid down to prepare for the derivation of a formula for the class of all ranges of partial enumerations. The general idea here is to write any partial enumeration as the co-restriction of **enum[x]** to an ordinal, starting with the known fact that any partial enumeration is the restriction of **enum[x]** to an ordinal. If that ordinal is in the domain of **enum[x]**, then applying **enum[x]** to that ordinal yields another ordinal that can be used to write the partial enumeration as a co-restriction to an ordinal.

Lemma. (Removing an **ord** wrapper.)

```
In[17]:= SubstTest[implies, equal[y, ord[t]],
           member[composite[id[y], enum[x]], ENUMS], t → y] // Reverse

Out[17]= or[member[composite[id[y], enum[x]], ENUMS], not[member[y, OMEGA]]] == True

In[18]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[19]:= SubstTest[implies, member[y, OMEGA],
           member[composite[id[y], enum[x]], ENUMS], y → APPLY[enum[x], ord[y]]] // Reverse

Out[19]= or[member[composite[id[APPLY[enum[x], ord[y]]], enum[x]], ENUMS],
           not[member[ord[y], domain[enum[x]]]]] == True

In[20]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[21]:= SubstTest[implies, and[member[u, ENUMS], member[v, ENUMS], equal[range[u], range[v]]],
           equal[u, v], {u -> composite[enum[x], id[ord[y]]],
            v -> composite[id[APPLY[enum[x], ord[y]]], enum[x]]}] // Reverse

Out[21]= or[equal[composite[enum[x], id[ord[y]]],
             composite[id[APPLY[enum[x], ord[y]]], enum[x]]],
           not[equal[image[enum[x], ord[y]], intersection[OMEGA, x, APPLY[enum[x], ord[y]]]]],
           not[member[composite[id[APPLY[enum[x], ord[y]]], enum[x]], ENUMS]]] == True

In[22]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. Rewriting a restriction of **enum[x]** to an ordinal as a co-restriction.

```
In[23]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
            not[implies[p1, p4]], {p1 → member[ord[y], domain[enum[x]]],
             p2 -> equal[image[enum[x], ord[y]], intersection[OMEGA, x, APPLY[enum[x], ord[y]]]],
             p3 → member[composite[id[APPLY[enum[x], ord[y]]], enum[x]], ENUMS],
             p4 -> equal[composite[enum[x], id[ord[y]]],
               composite[id[APPLY[enum[x], ord[y]]], enum[x]]]}] // Reverse

Out[23]= or[equal[composite[enum[x], id[ord[y]]],
             composite[id[APPLY[enum[x], ord[y]]], enum[x]]],
           not[member[ord[y], domain[enum[x]]]]] == True

In[24]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Corollary. (Remove the **ord** wrappers.)

```
In[25]:= SubstTest[implies, equal[y, ord[t]],
           or[equal[composite[enum[x], id[y]], composite[id[APPLY[enum[x], y]], enum[x]]],
            not[member[y, domain[enum[x]]]]], t → y] // Reverse

Out[25]= or[equal[composite[enum[x], id[y]], composite[id[APPLY[enum[x], y]], enum[x]]],
           not[member[y, domain[enum[x]]]]] == True

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

There are two cases to be considered, depending on whether or not the class of ordinals in **x** is a set or a proper class.

Lemma. (The case of a set.)

```
In[27]:= SubstTest[implies, and[member[u, OMEGA], member[v, OMEGA], subclass[u, v]],
           or[equal[u, v], member[u, v]], v → domain[enum[x]]] // Reverse

Out[27]= or[equal[u, domain[enum[x]]], member[u, domain[enum[x]]], not[member[u, OMEGA]],
           not[member[intersection[OMEGA, x], V]], not[subclass[u, domain[enum[x]]]]] == True

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

Lemma. (The case of a proper class.)

```
In[29]:= SubstTest[implies, and[member[u, OMEGA], equal[v, OMEGA], subclass[u, v]],
           member[u, v], v → domain[enum[x]]] // Reverse

Out[29]= or[member[u, domain[enum[x]]], member[intersection[OMEGA, x], V],
           not[member[u, OMEGA]], not[subclass[u, domain[enum[x]]]]] == True

In[30]:= (% /. {u → u_, x → x_}) /. Equal → SetDelayed
```

The next lemma combines the two cases considered in the preceding two lemmas.

Lemma.

```
In[31]:= Map[not, SubstTest[and, implies[p1, or[p2, p3, p4]],
            implies[and[p1, p3], p4], not[implies[p1, or[p2, p4]]],
            {p1 → and[member[u, OMEGA], subclass[u, domain[enum[x]]]],
             p2 → equal[u, domain[enum[x]]], p3 → not[member[intersection[OMEGA, x], V]],
             p4 → member[u, domain[enum[x]]]}]] // Reverse

Out[31]= or[equal[u, domain[enum[x]]], member[u, domain[enum[x]]],
           not[member[u, OMEGA]], not[subclass[u, domain[enum[x]]]]] == True

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

Lemma. A simplification rule.

```
In[33]:= ImageComp[IMAGE[id[x]], IMAGE[id[OMEGA]], OMEGA]

Out[33]= image[IMAGE[id[intersection[OMEGA, x]]], OMEGA] == image[IMAGE[id[x]], OMEGA]

In[34]:= image[IMAGE[id[intersection[OMEGA, x_]]], OMEGA] := image[IMAGE[id[x]], OMEGA]
```

Theorem. A condition for membership in the class **image[IMAGE[id[x]], Ω]**.

```
In[35]:= (SubstTest[or, member[intersection[z, t], image[IMAGE[id[z]], OMEGA]],
            not[member[t, OMEGA]], t → APPLY[enum[z], domain[t]]]) /.
          z → intersection[OMEGA, x] // Reverse

Out[35]= or[member[intersection[OMEGA, x, APPLY[enum[x], domain[t]]],
            image[IMAGE[id[x]], OMEGA]], not[member[domain[t], domain[enum[x]]]]] == True

In[36]:= (% /. {x → x_, t → t_}) /. Equal → SetDelayed
```

Technical Lemma (This is needed shortly to get around equality substitution limitations in the **GOEDEL** program.)

```
In[37]:= SubstTest[implies, equal[t, w], equal[range[t], range[w]],
           w -> composite[id[APPLY[enum[x], domain[t]]], enum[x]]] // Reverse

Out[37]= or[equal[intersection[OMEGA, x, APPLY[enum[x], domain[t]]], range[t]],
           not[equal[t, composite[id[APPLY[enum[x], domain[t]]], enum[x]]]]] == True

In[38]:= (% /. {x → x_, t → t_}) /. Equal → SetDelayed
```

---

breakup into two cases

Theorem. The two cases.

```
In[39]:= Map[not, SubstTest[and, implies[p1, p2],
            implies[p1, p3], implies[p2, p4], implies[and[p3, p4], or[p5, p6]],
            not[implies[p1, or[p5, p6]]], {p1 → member[t, partenum[x]],
             p2 → equal[t, composite[enum[x], id[domain[t]]]], p3 → member[domain[t], OMEGA],
             p4 → subclass[domain[t], domain[enum[x]]], p5 → member[domain[t], domain[enum[x]]],
             p6 → equal[domain[t], domain[enum[x]]]}]] // Reverse

Out[39]= or[equal[domain[t], domain[enum[x]]],
           member[domain[t], domain[enum[x]]], not[member[t, partenum[x]]]] == True

In[40]:= or[equal[domain[t_], domain[enum[x]]],
           member[domain[t_], domain[enum[x_]]], not[member[t_, partenum[x_]]]] := True
```

In the easy case that **domain[t] = domain[enum[x]]**, one has **t = enum[x]**.

Theorem.

```
In[41]:= Map[not, SubstTest[and, implies[p1, p3],
            implies[and[p1, p2, p3], p4], not[implies[and[p1, p2], p4]],
            {p1 -> member[t, partenum[x]], p2 -> equal[domain[t], domain[enum[x]]], p3 →
              equal[t, composite[enum[x], id[domain[t]]]], p4 → equal[t, enum[x]]}]] // Reverse

Out[41]= or[equal[t, enum[x]],
           not[equal[domain[t], domain[enum[x]]]], not[member[t, partenum[x]]]] == True

In[42]:= or[equal[t_, enum[x_]], not[equal[domain[t_], domain[enum[x_]]]],
           not[member[t_, partenum[x_]]]] := True
```

Lemma. A simplification rule.

```
In[43]:= equal[intersection[x, tc[x]], x]

Out[43]= True

In[44]:= intersection[x_, tc[x_]] := x
```

Lemma.

```
In[45]:= (SubstTest[implies, member[t, OMEGA], member[intersection[t, w],
             image[IMAGE[id[w]], OMEGA]], t → tc[w]] // Reverse) /. w → intersection[OMEGA, x]

Out[45]= or[member[intersection[OMEGA, x], image[IMAGE[id[x]], OMEGA]],
           not[member[intersection[OMEGA, x], V]]] == True

In[46]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. The easy case.

```
In[47]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[and[p1, p3], p4],
            not[implies[and[p1, p2], p4]], {p1 -> member[t, partenum[x]],
             p2 -> equal[domain[t], domain[enum[x]]], p3 → equal[t, enum[x]],
             p4 -> member[range[t], image[IMAGE[id[x]], OMEGA]]}]] // Reverse

Out[47]= or[member[range[t], image[IMAGE[id[x]], OMEGA]],
           not[equal[domain[t], domain[enum[x]]]], not[member[t, partenum[x]]]] == True

In[48]:= (% /. {x → x_, t → t_}) /. Equal → SetDelayed
```

the second case

The following proof step is omitted in the next lemma: **implies[and[p8, p9, p10], p11]**.

Lemma.

```
In[49]:= Map[not,
           SubstTest[and, implies[p1, p2], implies[and[p2, p5], p7], implies[and[p2, p7], p8],
            implies[p5, p9], implies[p8, p10], not[implies[and[p1, p5], p11]],
            {p1 → member[t, partenum[x]], p2 → equal[t, composite[enum[x], id[domain[t]]]],
             p5 → member[domain[t], domain[enum[x]]], p7 → equal[composite[enum[x],
                id[domain[t]]], composite[id[APPLY[enum[x], domain[t]]], enum[x]]],
             p8 -> equal[t, composite[id[APPLY[enum[x], domain[t]]], enum[x]]], p9 -> member[
               intersection[OMEGA, x, APPLY[enum[x], domain[t]]], image[IMAGE[id[x]], OMEGA]],
             p10 → equal[range[t], intersection[OMEGA, x, APPLY[enum[x], domain[t]]]],
             p11 → member[range[t], image[IMAGE[id[x]], OMEGA]]}]] // Reverse

Out[49]= or[member[range[t], image[IMAGE[id[x]], OMEGA]],
           not[member[t, partenum[x]]], not[member[domain[t], domain[enum[x]]]]] == True

In[50]:= (% /. {t → t_, x → x_}) /. Equal → SetDelayed
```

Combine the two cases.

Main Theorem. If **t** is a partial enumeration of the ordinals in **x**, then the range of **t** is the intersection of **x** with an ordinal.

```
In[51]:= Map[not, SubstTest[and, implies[p1, or[p5, p6]],
           not[implies[p1, p11]], {p1 -> member[t, partenum[x]],
            p5 → member[domain[t], domain[enum[x]]], p6 → equal[domain[t], domain[enum[x]]],
            p11 → member[range[t], image[IMAGE[id[x]], OMEGA]]}]] // Reverse

Out[51]= or[member[range[t], image[IMAGE[id[x]], OMEGA]], not[member[t, partenum[x]]]] == True

In[52]:= or[member[range[t_], image[IMAGE[id[x_]], OMEGA]],
           not[member[t_, partenum[x_]]]] := True
```

Theorem. Eliminate the variable **t**.

```
In[53]:= Map[equal[V, #] &,
          dif[partenum[x], image[inverse[IMAGE[SECOND]], image[IMAGE[id[x]], OMEGA]]] //
            complement // Normality]

Out[53]= subclass[image[IMAGE[SECOND], partenum[x]], image[IMAGE[id[x]], OMEGA]] == True

In[54]:= (% /. x → x_) /. Equal → SetDelayed
```

The reverse inclusion is already available.

Main Theorem. The class of ranges of the partial enumerations of a class **x** is the class of all intersections of **x** with ordinals.

```
In[55]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> image[IMAGE[SECOND], partenum[x]], v -> image[IMAGE[id[x]], OMEGA]}]

Out[55]= equal[image[IMAGE[SECOND], partenum[x]], image[IMAGE[id[x]], OMEGA]] == True

In[56]:= image[IMAGE[SECOND], partenum[x_]] := image[IMAGE[id[x]], OMEGA]
```

application: reify rules

In this final section the result of the preceding section is used to derive a **reify** rule for **partenum** and for **enum**.

Lemma. A simplification rule.

```
In[57]:= equal[intersection[partenum[x], ENUMS], partenum[x]]

Out[57]= True

In[58]:= intersection[ENUMS, partenum[x_]] := partenum[x]
```

Lemma. A simplification rule.

```
In[59]:= SubstTest[composite, funpart[t], inverse[funpart[t]],
           t -> composite[id[ENUMS], inverse[IMAGE[SECOND]]]] // Reverse

Out[59]= composite[id[ENUMS], inverse[IMAGE[SECOND]], IMAGE[SECOND], id[ENUMS]] == id[ENUMS]
```

```
In[60]:= composite[id[ENUMS], inverse[IMAGE[SECOND]], IMAGE[SECOND], id[ENUMS]] := id[ENUMS]
```

Theorem. A formula for **partenum[x]**.

```
In[61]:= ImageComp[composite[id[ENUMS], inverse[IMAGE[SECOND]]],
           composite[IMAGE[SECOND], id[ENUMS]], partenum[x]] // Reverse

Out[61]= intersection[ENUMS, image[inverse[IMAGE[SECOND]], image[IMAGE[id[x]], OMEGA]]] ==
          partenum[x]

In[62]:= intersection[ENUMS,
           image[inverse[IMAGE[SECOND]], image[IMAGE[id[x_]], OMEGA]]] := partenum[x]
```

Theorem. A **reify rule** for **partenum**.

```
In[63]:= SubstTest[reify, x, intersection[t,
            image[inverse[IMAGE[SECOND]], image[IMAGE[id[f[x]]], OMEGA]]], t → ENUMS] // Reverse

Out[63]= reify[x, partenum[f[x]]] == composite[id[ENUMS], inverse[IMAGE[SECOND]], IMAGE[SECOND],
           IMAGE[id[reify[x, f[x]]]], CART, id[cart[V, OMEGA]], inverse[FIRST], SINGLETON]

In[64]:= reify[x_, partenum[y_]] := composite[id[ENUMS], inverse[IMAGE[SECOND]], IMAGE[SECOND],
           IMAGE[id[reify[x, y]]], CART, id[cart[V, OMEGA]], inverse[FIRST], SINGLETON]
```

Corollary. A **reify rule** for **enum**.

```
In[65]:= SubstTest[reify, x, image[t, partenum[f[x]]], t → inverse[E]] // Reverse

Out[65]= reify[x, enum[f[x]]] ==
          composite[inverse[E], id[ENUMS], inverse[IMAGE[SECOND]], IMAGE[SECOND],
           IMAGE[id[reify[x, f[x]]]], CART, id[cart[V, OMEGA]], inverse[FIRST], SINGLETON]

In[66]:= reify[x_, enum[y_]] :=
          composite[inverse[E], id[ENUMS], inverse[IMAGE[SECOND]], IMAGE[SECOND],
           IMAGE[id[reify[x, y]]], CART, id[cart[V, OMEGA]], inverse[FIRST], SINGLETON]
```