ordinal addition, part 5.

Johan G. F. Belinfante
2010 December 17

```
In[1]:= SetDirectory["l:"]; << goedel.10dec16b

        :Package Title: goedel.10dec16b            2010 December 16 at 6:20 p.m.

        It is now:  2010 Dec 17 at 20:3

        Loading Simplification Rules

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

        weightlimit = 40
```

summary

This fifth notebook in a series about ordinal addition is concerned with a strict monotonicity property of ordinal addition. Some results are derived first using **ord** wrappers, and later generalized, removing the **ord** wrappers.

monotonicity properties of ordplus[x]

Both **ordplus[x]** and its inverse are monotone with respect to membership.

Theorem.

```
In[2]:= SubstTest[subclass, enum[t], composite[S, IMAGE[enum[t]]], t →
           intersection[OMEGA, complement[x], image[V, intersection[OMEGA, set[x]]]]] // Reverse

Out[2]= subclass[ordplus[x], composite[S, IMAGE[ordplus[x]]]] == True

In[3]:= subclass[ordplus[x_], composite[S, IMAGE[ordplus[x_]]]] := True
```

Theorem.

```
In[4]:= SubstTest[subclass, inverse[enum[t]], composite[S, IMAGE[inverse[enum[t]]], t →
           intersection[OMEGA, complement[x], image[V, intersection[OMEGA, set[x]]]]] // Reverse

Out[4]= subclass[inverse[ordplus[x]], composite[S, IMAGE[inverse[ordplus[x]]]]] == True

In[5]:= subclass[inverse[ordplus[x_]], composite[S, IMAGE[inverse[ordplus[x_]]]]] := True
```

Theorem.

```
In[6]:= SubstTest[subclass, enum[t], S, t ->
          intersection[OMEGA, complement[x], image[V, intersection[OMEGA, set[x]]]]] // Reverse

Out[6]= subclass[ordplus[x], S] == True

In[7]:= subclass[ordplus[x_], S] := True
```

---

monotonicity of ordadd[x, –]

In this section a rewrite rule is derived about monotonicity of **ordadd[x, –]** when **x** is fixed. Since there are three variables and each may or may not be an ordinal, many lemmas are needed to cover all possible cases.

Lemma. A result obtained by specializing a general theorem about **enum[x]**.

```
In[8]:= SubstTest[implies, and[member[y, z], member[z, domain[enum[t]]]],
          member[APPLY[enum[t], y], APPLY[enum[t], z]], t ->
           intersection[OMEGA, complement[x], image[V, intersection[OMEGA, set[x]]]]] // Reverse

Out[8]= or[member[ordadd[x, y], ordadd[x, z]],
          not[member[x, OMEGA]], not[member[y, z]], not[member[z, OMEGA]]] == True

In[9]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma. If **ordadd[x, y]** is a set, then **x** is an ordinal.

```
In[10]:= Map[or[member[x, OMEGA], #] &,
           SubstTest[implies, member[t, z], member[t, V], t → ordadd[x, y]]] // Reverse

Out[10]= or[member[x, OMEGA], not[member[ordadd[x, y], z]]] == True

In[11]:= or[member[x_, OMEGA], not[member[ordadd[x_, y_], z_]]] := True
```

Lemma. If **ordadd[x, y]** is a set, then **y** is an ordinal.

```
In[12]:= Map[or[member[y, OMEGA], #] &,
           SubstTest[implies, member[t, z], member[t, V], t → ordadd[x, y]]] // Reverse

Out[12]= or[member[y, OMEGA], not[member[ordadd[x, y], z]]] == True

In[13]:= or[member[y_, OMEGA], not[member[ordadd[x_, y_], z_]]] := True
```

Theorem. The case that **x** and **y** are ordinals but **z** is not.

```
In[14]:= SubstTest[implies, and[member[u, v], equal[v, w]],
           member[u, w], {u → ordadd[x, y], v → V, w -> ordadd[x, z]}] // Reverse

Out[14]= or[member[z, OMEGA], member[ordadd[x, y], ordadd[x, z]],
           not[member[x, OMEGA]], not[member[y, OMEGA]]] == True

In[15]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma. A result obtained by specializing a general theorem about **enum[x]**. (This has a redundant literal)

```
In[16]:= SubstTest[or, member[y, z], not[member[z, domain[enum[t]]]],
           not[member[APPLY[enum[t], y], APPLY[enum[t], z]]], t -> intersection[
             OMEGA, complement[x], image[V, intersection[OMEGA, set[x]]]]] // Reverse

Out[16]= or[member[y, z], not[member[x, OMEGA]],
           not[member[z, OMEGA]], not[member[ordadd[x, y], ordadd[x, z]]]] == True

In[17]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Theorem. (Removing a redundant literal from the result obtained in the preceding lemma.

```
In[18]:= SubstTest[and, implies[p, q], or[p, q], {p → member[x, OMEGA],
           q -> or[member[y, z], not[member[z, OMEGA]], not[member[ordadd[x, y], ordadd[x, z]]]]}]

Out[18]= or[member[y, z], not[member[z, OMEGA]], not[member[ordadd[x, y], ordadd[x, z]]]] == True

In[19]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Main Theorem. A general rewrite rule about strict monotonicity of **ordadd[x, –]** for fixed **x**.

```
In[20]:= equiv[member[ordadd[x, y], ordadd[x, z]],
             or[and[member[x, OMEGA], member[y, OMEGA], not[member[z, OMEGA]]],
              and[member[x, OMEGA], member[y, z], member[z, OMEGA]]]] // not // not

Out[20]= True

In[21]:= member[ordadd[x_, y_], ordadd[x_, z_]] :=
           or[and[member[x, OMEGA], member[y, OMEGA], not[member[z, OMEGA]]],
            and[member[x, OMEGA], member[y, z], member[z, OMEGA]]]
```

corollaries

Corollary. A case where one of the variables is **0**.

```
In[22]:= SubstTest[member, ordadd[x, y], ordadd[x, t], t → 0] // Reverse

Out[22]= member[ordadd[x, y], x] == False

In[23]:= member[ordadd[x_, y_], x_] := False
```

A similar result will be obtained using **subclass** in place of **member**. Inclusions for ordinals can be restated as negated reversed membership. For this it is convenient to introduce **ord** wrappers.

Corollary.

```
In[24]:= SubstTest[subclass, ord[x], ord[t], t → ordadd[ord[x], ord[y]]] // Reverse

Out[24]= subclass[ord[x], ordadd[ord[x], ord[y]]] == True
```

```
In[25]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. (Removing the two **ord** wrappers.)

```
In[26]:= SubstTest[implies, and[equal[x, ord[u]], equal[y, ord[v]]],
           subclass[x, ordadd[x, y]], {u → x, v → y}] // Reverse
Out[26]= or[not[member[x, OMEGA]], not[member[y, OMEGA]], subclass[x, ordadd[x, y]]] == True

In[27]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. (Needed to eliminate two redundant literals in the preceding lemma.)

```
In[28]:= SubstTest[implies, equal[v, V], subclass[w, v], v → ordadd[x, y]] // Reverse // MapNotNot
Out[28]= or[and[member[x, OMEGA], member[y, OMEGA]], subclass[w, ordadd[x, y]]] == True

In[29]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

The above two lemmas can be combined, eliminating two redundant literals.

Theorem.

```
In[30]:= SubstTest[and, implies[p, q], or[p, q],
           {p -> and[member[x, OMEGA], member[y, OMEGA]], q -> subclass[x, ordadd[x, y]]}]
Out[30]= subclass[x, ordadd[x, y]] == True

In[31]:= subclass[x_, ordadd[x_, y_]] := True
```