ordinal addition, part 3.

Johan G. F. Belinfante
2010 December 16

```
In[1]:= SetDirectory["l:"]; << goedel.10dec15b

        :Package Title: goedel.10dec15b              2010 December 15 at 7:10 p.m.

        It is now:  2010 Dec 16 at 14:21

        Loading Simplification Rules

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

        weightlimit = 40
```

summary

In this third notebook about ordinal addition, the binary constructors **ordadd[x, y]** and **ordsub[x, y]** are introduced. Membership and function application rules for **ORDADD** and **ordplus[x]** are stated in terms of these constructors.

ordinal addition

The constructor **ordadd[x, y]** is defined for any two classes **x** and **y**, not just for ordinals.

Definition.

```
In[2]:= APPLY[ORDADD, PAIR[x_, y_]] := ordadd[x, y]
```

Theorem. Adding two ordinals yields another ordinal.

```
In[3]:= SubstTest[member, APPLY[funpart[t], w],
          range[funpart[t]], {t → ORDADD, w→ PAIR[x, y]}] // Reverse
Out[3]= member[ordadd[x, y], OMEGA] == and[member[x, OMEGA], member[y, OMEGA]]

In[4]:= member[ordadd[x_, y_], OMEGA] := and[member[x, OMEGA], member[y, OMEGA]]
```

Theorem. Sethood for **ordadd[x,y]**.

```
In[5]:= SubstTest[member, APPLY[t, PAIR[x, y]], V, t → ORDADD] // Reverse
Out[5]= member[ordadd[x, y], V] == and[member[x, OMEGA], member[y, OMEGA]]

In[6]:= member[ordadd[x_, y_], V] := and[member[x, OMEGA], member[y, OMEGA]]
```

Theorem. If either **x** or **y** is not an ordinal, then **ordadd[x, y] = V**.

```
In[7]:= SubstTest[equal, APPLY[funpart[t], w], V, {t → ORDADD, w → PAIR[x, y]}] // Reverse

Out[7]= equal[V, ordadd[x, y]] == or[not[member[x, OMEGA]], not[member[y, OMEGA]]]

In[8]:= equal[V, ordadd[x_, y_]] := or[not[member[x, OMEGA]], not[member[y, OMEGA]]]
```

vertical section and APPLY rules for ORDADD and ordplus[x]

Theorem. Vertical section rule for **ORDADD**.

```
In[9]:= SubstTest[image, funpart[t], set[PAIR[x, y]], t → ORDADD] // Reverse

Out[9]= image[ORDADD, cart[set[x], set[y]]] == set[ordadd[x, y]]

In[10]:= image[ORDADD, cart[set[x_], set[y_]]] := set[ordadd[x, y]]
```

Theorem. A function application rule for **ordplus[x]**.

```
In[11]:= ApComp[ORDADD, LEFT[x], y] // Reverse

Out[11]= APPLY[ordplus[x], y] == ordadd[x, y]

In[12]:= APPLY[ordplus[x_], y_] := ordadd[x, y]
```

Theorem. A vertical section rule for **ordplus[x]**.

```
In[13]:= SubstTest[image, funpart[t], set[y], t → ordplus[x]] // Reverse

Out[13]= image[ordplus[x], set[y]] == set[ordadd[x, y]]

In[14]:= image[ordplus[x_], set[y_]] := set[ordadd[x, y]]
```

membership rules for ORDADD and ordplus[x]

Lemma. Temporary simplification rule.

```
In[15]:= SubstTest[member, pair[pair[x, y], z], composite[t, id[cart[V, V]]], t → ORDADD]

Out[15]= and[member[x, V], member[y, V], member[pair[pair[x, y], z], ORDADD]] ==
          member[pair[pair[x, y], z], ORDADD]

In[16]:= and[member[x_, V], member[y_, V], member[pair[pair[x_, y_], z_], ORDADD]] :=
          member[pair[pair[x, y], z], ORDADD]
```

Theorem. Membership rule for **ORDADD**.

```
In[17]:= SubstTest[member, z, image[t, set[PAIR[x, y]]], t → ORDADD]
Out[17]= member[pair[pair[x, y], z], ORDADD] == and[equal[z, ordadd[x, y]], member[z, V]]

In[18]:= member[pair[pair[x_, y_], z_], ORDADD] := and[equal[z, ordadd[x, y]], member[z, V]]
```

Theorem. Membership rule for **ordplus[x]**.

```
In[19]:= SubstTest[member, y, image[t, set[x]], t → ordplus[z]]
Out[19]= member[pair[x, y], ordplus[z]] == and[equal[y, ordadd[z, x]], member[y, V]]

In[20]:= member[pair[x_, y_], ordplus[z_]] := and[equal[y, ordadd[z, x]], member[y, V]]
```

---

ordinal subtraction

The constructor **ordsub[x, y]** is also defined for any two classes **x** and **y**.

Definition.

```
In[21]:= APPLY[rotate[ORDADD], PAIR[x_, y_]] := ordsub[x, y]
```

Lemma. A simplification rule.

```
In[22]:= composite[FIRST, inverse[ORDADD]] // DoubleInverse
Out[22]= composite[FIRST, inverse[ORDADD]] == composite[id[OMEGA], inverse[S], id[OMEGA]]

In[23]:= composite[FIRST, inverse[ORDADD]] := composite[id[OMEGA], inverse[S], id[OMEGA]]
```

Theorem. Subtracting an ordinal **y** from an ordinal **x** only yields an ordinal when **y ⊂ x**.

```
In[24]:= SubstTest[member, APPLY[funpart[t], w],
           range[funpart[t]], {t → rotate[ORDADD], w → PAIR[x, y]}] // Reverse
Out[24]= member[ordsub[x, y], OMEGA] == and[member[x, OMEGA], member[y, OMEGA], subclass[y, x]]

In[25]:= member[ordsub[x_, y_], OMEGA] := and[member[x, OMEGA], member[y, OMEGA], subclass[y, x]]
```

Theorem. Sethood rule for **ordsub[x,y]**.

```
In[26]:= SubstTest[member, APPLY[t, PAIR[x, y]], V, t → rotate[ORDADD]] // Reverse
Out[26]= member[ordsub[x, y], V] == and[member[x, OMEGA], member[y, OMEGA], subclass[y, x]]

In[27]:= member[ordsub[x_, y_], V] := and[member[x, OMEGA], member[y, OMEGA], subclass[y, x]]
```

Theorem. When **ordsub[x, y]** is not an ordinal, it is equal to **V**.

```
In[28]:= SubstTest[equal, APPLY[funpart[t], w], V, {t → rotate[ORDADD], w → PAIR[x, y]}] // Reverse

Out[28]= equal[V, ordsub[x, y]] ==
          or[not[member[x, OMEGA]], not[member[y, OMEGA]], not[subclass[y, x]]]

In[29]:= equal[V, ordsub[x_, y_]] :=
          or[not[member[x, OMEGA]], not[member[y, OMEGA]], not[subclass[y, x]]]
```

vertical section and APPLY rules for rotate[ORDADD] and inverse[ordplus[x]]

Theorem. Vertical section rule for **rotate[ORDADD]**.

```
In[30]:= SubstTest[image, funpart[t], set[PAIR[x, y]], t → rotate[ORDADD]] // Reverse

Out[30]= image[image[inverse[ORDADD], set[x]], set[y]] == set[ordsub[x, y]]

In[31]:= image[image[inverse[ORDADD], set[x_]], set[y_]] := set[ordsub[x, y]]
```

Theorem. Function application rule for **inverse[ordplus[x]]**.

```
In[32]:= ApComp[rotate[ORDADD], RIGHT[x], y] // Reverse

Out[32]= APPLY[inverse[ordplus[x]], y] == ordsub[y, x]

In[33]:= APPLY[inverse[ordplus[x_]], y_] := ordsub[y, x]
```

Theorem. Vertical section rule for **inverse[ordplus[x]]**.

```
In[34]:= SubstTest[image, funpart[t], set[y], t → inverse[ordplus[x]]] // Reverse

Out[34]= image[inverse[ordplus[x]], set[y]] == set[ordsub[y, x]]

In[35]:= image[inverse[ordplus[x_]], set[y_]] := set[ordsub[y, x]]
```

relating ordinal addition and subtraction

The connection between ordinal addition and ordinal subtraction is the topic in this section. The results are complicated because **x** and **y** are allowed to be arbitrary classes.

Theorem. A general formula for **x + (y – x)**.

```
In[36]:= ApComp[ordplus[x], inverse[ordplus[x]], y]

Out[36]= ordadd[x, ordsub[y, x]] == union[y, complement[image[V, intersection[OMEGA, set[x]]]],
           complement[image[V, intersection[OMEGA, set[y]]]], image[V, intersection[x, set[y]]]]
```

```
In[37]:= ordadd[x_, ordsub[y_, x_]] :=
          union[y, complement[image[V, intersection[OMEGA, set[x]]]],
           complement[image[V, intersection[OMEGA, set[y]]]], image[V, intersection[x, set[y]]]]
```

This result looks better when **ord** wrappers are used.

```
In[38]:= equal[ordadd[ord[x], ordsub[ord[y], ord[x]]], ord[y]]

Out[38]= not[member[ord[y], ord[x]]]
```

Theorem. A general formula for **(x + y) – x**.

```
In[39]:= ApComp[inverse[ordplus[x]], ordplus[x], y]

Out[39]= ordsub[ordadd[x, y], x] == union[y, complement[image[V, intersection[OMEGA, set[x]]]],
           complement[image[V, intersection[OMEGA, set[y]]]]]
```

```
In[40]:= ordsub[ordadd[x_, y_], x_] :=
          union[y, complement[image[V, intersection[OMEGA, set[x]]]],
           complement[image[V, intersection[OMEGA, set[y]]]]]
```

With **ord** wrappers this reduces to the following.

```
In[41]:= ordsub[ordadd[ord[x], ord[y]], ord[x]]

Out[41]= ord[y]
```

adding and subtracting 0

In this section formulas for adding and subtracting 0 are derived.

Theorem. A variable-free result.

```
In[42]:= SubstTest[enum, complement[ord[x]], x → 0]

Out[42]= ordplus[0] == id[OMEGA]
```

```
In[43]:= ordplus[0] := id[OMEGA]
```

When one introduces a variable **x**, the result becomes more complicated because **x** need not be an ordinal.

Theorem. A formula for **0 + x**.

```
In[44]:= SubstTest[APPLY, ordplus[t], x, t → 0]

Out[44]= ordadd[0, x] == union[x, complement[image[V, intersection[OMEGA, set[x]]]]]
```

```
In[45]:= ordadd[0, x_] := union[x, complement[image[V, intersection[OMEGA, set[x]]]]]
```

Theorem. A formula for **x – 0**.

```
In[46]:= SubstTest[APPLY, inverse[ordplus[t]], x, t → 0]
Out[46]= ordsub[x, 0] == union[x, complement[image[V, intersection[OMEGA, set[x]]]]]

In[47]:= ordsub[x_, 0] := union[x, complement[image[V, intersection[OMEGA, set[x]]]]]
```

right addition by 0

A formula for **x + 0** is derived in this section.

Lemma. The special case that **x** is wrapped with **ord**.

```
In[48]:= SubstTest[APPLY, enum[t], 0, t → complement[ord[x]]] // Reverse
Out[48]= ordadd[ord[x], 0] == ord[x]

In[49]:= ordadd[ord[x_], 0] := ord[x]
```

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

```
In[50]:= SubstTest[implies, equal[x, ord[t]], equal[ordadd[x, 0], x], t → x] // Reverse
Out[50]= or[equal[x, ordadd[x, 0]], not[member[x, OMEGA]]] == True

In[51]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. A general formula for **x + 0**.

```
In[52]:= equal[ordadd[x, 0], union[x, complement[image[V, intersection[OMEGA, set[x]]]]]]
Out[52]= True

In[53]:= ordadd[x_, 0] := union[x, complement[image[V, intersection[OMEGA, set[x]]]]]
```

Corollary. A variable-free version for ordinal right-addition by **0**.

```
In[54]:= composite[ORDADD, RIGHT[0]] // RelnNormality
Out[54]= composite[ORDADD, RIGHT[0]] == id[OMEGA]

In[55]:= composite[ORDADD, RIGHT[0]] := id[OMEGA]
```

a left-cancellation law

Lemma. If both **x + y** and **x + z** are equal to **V**, then **x + y = x + z**.

```
In[56]:= SubstTest[implies, and[equal[u, v], equal[v, w]],
           equal[u, w], {u → ordadd[x, y], v → V, w → ordadd[x, z]}] // Reverse

Out[56]= or[and[member[x, OMEGA], member[y, OMEGA]],
           and[member[x, OMEGA], member[z, OMEGA]], equal[ordadd[x, y], ordadd[x, z]]] == True

In[57]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma. If **x** and **y** are ordinals and **z** is not, the **x + y** cannot be equal to **x + z**.

```
In[58]:= SubstTest[implies, and[member[u, OMEGA], equal[v, V]],
           not[equal[u, v]], {u → ordadd[x, y], v → ordadd[x, z]}] // Reverse

Out[58]= or[member[z, OMEGA], not[equal[ordadd[x, y], ordadd[x, z]]],
           not[member[x, OMEGA]], not[member[y, OMEGA]]] == True

In[59]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma. If all three variables are ordinals, then **x + y = x + z** implies **y = z**.

```
In[60]:= Map[implies[and[member[y, OMEGA], member[z, OMEGA]], #] &,
           SubstTest[implies, equal[u, v], equal[ordsub[u, x], ordsub[v, x]],
            {u → ordadd[x, y], v → ordadd[x, z]}]] // Reverse

Out[60]= or[equal[y, z], not[equal[ordadd[x, y], ordadd[x, z]]],
           not[member[x, OMEGA]], not[member[y, OMEGA]], not[member[z, OMEGA]]] == True

In[61]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

The above lemma has a redundant literal.

Lemma. (Removing a redundant literal.)

```
In[62]:= SubstTest[and, implies[p, q], or[p, q],
          {p -> member[z, OMEGA], q -> or[equal[y, z], not[equal[ordadd[x, y], ordadd[x, z]]],
             not[member[x, OMEGA]], not[member[y, OMEGA]]]}]

Out[62]= or[equal[y, z], not[equal[ordadd[x, y], ordadd[x, z]]],
           not[member[x, OMEGA]], not[member[y, OMEGA]]] == True

In[63]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Theorem. The general case for the left cancellation law.

```
In[64]:= equiv[equal[ordadd[x, y], ordadd[x, z]],
            or[and[not[member[y, OMEGA]], not[member[z, OMEGA]]],
             equal[y, z], not[member[x, OMEGA]]]] // not // not

Out[64]= True

In[65]:= equal[ordadd[x_, y_], ordadd[x_, z_]] := or[
           and[not[member[y, OMEGA]], not[member[z, OMEGA]]], equal[y, z], not[member[x, OMEGA]]]
```

Comment. The left cancellation law simplifies as follows when **ord** wrappers are used.

```
In[66]:= equal[ordadd[ord[x], ord[y]], ordadd[ord[x], ord[z]]]

Out[66]= equal[ord[y], ord[z]]
```

a formula for x - x

The expression **x – x** is either **0** or **V** depending on whether or not the class **x** is an ordinal.

Lemma. The special case that **x** is wrapped with **ord**.

```
In[69]:= SubstTest[ordsub, ordadd[ord[x], y], ord[x], y → 0] // Reverse

Out[69]= ordsub[ord[x], ord[x]] == 0

In[70]:= (% /. x → x_) /. Equal → SetDelayed
```

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

```
In[71]:= SubstTest[implies, equal[x, ord[t]], equal[0, ordsub[x, x]], t → x] // Reverse

Out[71]= or[equal[0, ordsub[x, x]], not[member[x, OMEGA]]] == True

In[72]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. A general formula for **x – x**.

```
In[73]:= equal[ordsub[x, x], complement[image[V, intersection[OMEGA, set[x]]]]]

Out[73]= True

In[75]:= ordsub[x_, x_] := complement[image[V, intersection[OMEGA, set[x]]]]
```