ordinal addition, part 6

Johan G. F. Belinfante
2010 December 18

```
In[1]:= SetDirectory["l:"]; << goedel.10dec17a

        :Package Title: goedel.10dec17a           2010 December 17 at 8:15 p.m.

        It is now:  2010 Dec 18 at 12:47

        Loading Simplification Rules

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

        weightlimit = 40
```

summary

The main result in this sixth notebook in a series about ordinal addition is the associative law for ordinal addition. The key idea in the derivation is to make use of the fact that the composite of two enumerations is an enumeration.

a simplification rule

A simplification rule is derived here that is needed in the next section.

Lemma.

```
In[9]:= SubstTest[implies, member[t, OMEGA], subclass[t, OMEGA], t → ordadd[x, y]] // Reverse
Out[9]= or[not[member[x, OMEGA]], not[member[y, OMEGA]], subclass[ordadd[x, y], OMEGA]] == True

In[10]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[11]:= SubstTest[implies, equal[t, V], not[subclass[t, OMEGA]], t → ordadd[x, y]] // Reverse
Out[11]= or[and[member[x, OMEGA], member[y, OMEGA]], not[subclass[ordadd[x, y], OMEGA]]] == True

In[12]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. A simplification rule.

```
In[13]:= equiv[subclass[ordadd[x, y], OMEGA], and[member[x, OMEGA], member[y, OMEGA]]]
Out[13]= True
```

```
In[15]:= subclass[ordadd[x_, y_], OMEGA] := and[member[x, OMEGA], member[y, OMEGA]]
```

derivation using ord wrappers

It is convenient to begin by using **ord** wrappers, and to remove them later.

Lemma. Simplification rule.

```
In[16]:= equal[intersection[OMEGA, ordadd[ord[x], ord[y]]], ordadd[ord[x], ord[y]]]

Out[16]= True

In[18]:= intersection[OMEGA, ordadd[ord[x_], ord[y_]]] := ordadd[ord[x], ord[y]]
```

Theorem. A formula for the image of an ordinal under ordinal left-addition.

```
In[19]:= SubstTest[implies, member[v, domain[enum[u]]],
           equal[image[enum[u], v], intersection[APPLY[enum[u], v], u, OMEGA]],
           {u → complement[ord[x]], v → ord[y]}] // Reverse

Out[19]= equal[image[ordplus[ord[x]], ord[y]],
           intersection[complement[ord[x]], ordadd[ord[x], ord[y]]]] == True

In[20]:= image[ordplus[ord[x_]], ord[y_]] :=
          intersection[complement[ord[x]], ordadd[ord[x], ord[y]]]
```

Lemma. Simplification rule.

```
In[21]:= equal[intersection[complement[x], complement[ordadd[x, y]]], complement[ordadd[x, y]]]

Out[21]= True

In[22]:= intersection[complement[x_], complement[ordadd[x_, y_]]] := complement[ordadd[x, y]]
```

Lemma. A temporary formula for the composite of two left-additions, with **ord** wrappers.

```
In[23]:= SubstTest[equal, composite[enum[u], enum[v]], enum[image[enum[u], v]],
           {u → complement[ord[x]], v → complement[ord[y]]}] // Reverse

Out[23]= equal[composite[ordplus[ord[x]], ordplus[ord[y]]],
           ordplus[ordadd[ord[x], ord[y]]]] == True

In[24]:= composite[ordplus[ord[x_]], ordplus[ord[y_]]] := ordplus[ordadd[ord[x], ord[y]]]
```

Lemma. A temporary version of the associative law for ordinal addition using **ord** wrappers.

```
In[25]:= ApComp[ordplus[ord[x]], ordplus[ord[y]], ord[z]]

Out[25]= ordadd[ord[x], ordadd[ord[y], ord[z]]] == ordadd[ordadd[ord[x], ord[y]], ord[z]]

In[26]:= ordadd[ord[x_], ordadd[ord[y_], ord[z_]]] := ordadd[ordadd[ord[x], ord[y]], ord[z]]
```

removing the ord wrappers

In this section the results of the preceding one are improved by removing the **ord** wrappers, and a variable-free statement of the associative law is derived.

Lemma.

```
In[27]:= SubstTest[implies, and[equal[u, v], equal[v, w]], equal[u, w],
           {u -> ordadd[x, ordadd[y, z]], v → V, w -> ordadd[ordadd[x, y], z]}] // Reverse

Out[27]= or[and[member[x, OMEGA], member[y, OMEGA], member[z, OMEGA]],
           equal[ordadd[x, ordadd[y, z]], ordadd[ordadd[x, y], z]]] == True

In[28]:= (% /. {x → x_, y → y_, z → z_}) /. Equal→ SetDelayed
```

Lemma.

```
In[29]:= SubstTest[implies, and[equal[x, ord[u]], equal[y, ord[v]], equal[z, ord[w]]],
           equal[ordadd[x, ordadd[y, z]], ordadd[ordadd[x, y], z]], {u → x, v → y, w → z}] // Reverse

Out[29]= or[equal[ordadd[x, ordadd[y, z]], ordadd[ordadd[x, y], z]],
           not[member[x, OMEGA]], not[member[y, OMEGA]], not[member[z, OMEGA]]] == True

In[30]:= (% /. {x → x_, y → y_, z → z_}) /. Equal→ SetDelayed
```

The above two lemmas can now be combined to obtain a statement of the associative law free of **ord** wrappers.

Main Theorem. Associative law for ordinal addition.

```
In[31]:= SubstTest[and, implies[p, q], or[p, q],
          {p -> and[member[x, OMEGA], member[y, OMEGA], member[z, OMEGA]],
           q -> equal[ordadd[x, ordadd[y, z]], ordadd[ordadd[x, y], z]]}]

Out[31]= equal[ordadd[x, ordadd[y, z]], ordadd[ordadd[x, y], z]] == True

In[32]:= ordadd[x_, ordadd[y_, z_]] := ordadd[ordadd[x, y], z]
```

To obtain a variable-free statement, it suffices to derive an inclusion because any subclass of a function is a restriction. It is convenient to use **reify** to remove the variables.

Lemma. Elimination of all variables.

```
In[33]:= SubstTest[reify, x,
          dif[image[u, set[PAIR[PAIR[first[first[x]], second[first[x]]], second[x]]]],
           image[v, set[PAIR[PAIR[first[first[x]], second[first[x]]], second[x]]]],
          {u -> composite[ORDADD, cross[Id, ORDADD], ASSOC],
           v -> composite[ORDADD, cross[ORDADD, Id]]}]

Out[33]= composite[intersection[composite[complement[ORDADD], cross[ORDADD, Id]],
            composite[ORDADD, cross[Id, ORDADD], ASSOC]], id[cart[cart[V, V], V]]] == 0
```

```
In[34]:= % /. Equal → SetDelayed
```

Lemma. An inclusion.

```
In[35]:= SubstTest[empty, composite[dif[u, v], id[w]],
           {u -> composite[ORDADD, cross[Id, ORDADD], ASSOC],
            v -> composite[ORDADD, cross[ORDADD, Id]], w -> cart[cart[V, V], V]}]

Out[35]= subclass[composite[ORDADD, cross[Id, ORDADD], ASSOC],
           composite[ORDADD, cross[ORDADD, Id]]] == True
```

```
In[36]:= % /. Equal → SetDelayed
```

Lemma. A simplification rule.

```
In[37]:= composite[ORDADD, cross[ORDADD, id[OMEGA]]] // TripleRotate

Out[37]= composite[ORDADD, cross[ORDADD, id[OMEGA]]] == composite[ORDADD, cross[ORDADD, Id]]
```

```
In[38]:= % /. Equal → SetDelayed
```

Theorem. A variable-free statement of the associative law for ordinal addition.

```
In[39]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
           equal[u, composite[v, id[domain[u]]]],
           {u -> composite[ORDADD, cross[Id, ORDADD], ASSOC],
            v -> composite[ORDADD, cross[ORDADD, Id]]}] // Reverse

Out[39]= equal[composite[ORDADD, cross[ORDADD, Id]],
           composite[ORDADD, cross[Id, ORDADD], ASSOC]] == True
```

```
In[40]:= composite[ORDADD, cross[Id, ORDADD], ASSOC] := composite[ORDADD, cross[ORDADD, Id]]
```

Corollary.

```
In[41]:= SubstTest[implies, equal[composite[t, cross[t, Id]], composite[t, cross[Id, t], ASSOC]],
           associative[composite[t, id[cart[V, V]]]], t → ORDADD] // Reverse

Out[41]= associative[ORDADD] == True
```

```
In[42]:= associative[ORDADD] := True
```

corollaries about left and right addition

In this section, some rewrite rules about left and right addition are derived using the associative law for ordinal addition. In particular, most of the results of an earlier section can now be rederived sans the **ord** wrappers.

Theorem. The product of left-additions is the left-addition of the sum.

```
In[43]:= Assoc[composite[ORDADD, cross[Id, ORDADD]], ASSOC, LEFT[PAIR[x, y]]]

Out[43]= composite[ordplus[x], ordplus[y]] == ordplus[ordadd[x, y]]

In[44]:= composite[ordplus[x_], ordplus[y_]] := ordplus[ordadd[x, y]]
```

Another consequence of associativity will now be derived for left-addition that involves only one variable.

Lemma.

```
In[45]:= SubstTest[reify, x, dif[image[u, set[PAIR[first[x], second[x]]]],
           image[v, set[PAIR[first[x], second[x]]]]],
          {u -> composite[ordplus[x], ORDADD], v -> composite[ORDADD, cross[ordplus[x], Id]]}]

Out[45]= composite[intersection[composite[complement[ORDADD], cross[ordplus[x], Id]],
           composite[ordplus[x], ORDADD]], id[cart[V, V]]] == 0

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

Lemma. An inclusion.

```
In[47]:= SubstTest[empty, composite[dif[u, v], id[cart[V, V]]],
          {u → composite[ordplus[x], ORDADD], v -> composite[ORDADD, cross[ordplus[x], Id]]}]

Out[47]= subclass[composite[ordplus[x], ORDADD],
           composite[ORDADD, cross[ordplus[x], Id]]] == True

In[48]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. A simplification rule.

```
In[49]:= composite[ORDADD, cross[ordplus[x],
           id[intersection[OMEGA, image[V, intersection[OMEGA, set[x]]]]]]] // TripleRotate

Out[49]= composite[ORDADD,
           cross[ordplus[x], id[intersection[OMEGA, image[V, intersection[OMEGA, set[x]]]]]] ==
          composite[ORDADD, cross[ordplus[x], Id]]

In[50]:= composite[ORDADD, cross[ordplus[x_],
           id[intersection[OMEGA, image[V, intersection[OMEGA, set[x_]]]]]] :=
          composite[ORDADD, cross[ordplus[x], Id]]
```

The orientation of the following rewrite rule is based on an analogy with a similar rule for addition of natural numbers.

Theorem.

```
In[51]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
           equal[u, composite[v, id[domain[u]]]], {u -> composite[ordplus[x], ORDADD],
            v -> composite[ORDADD, cross[ordplus[x], Id]]}] // Reverse

Out[51]= equal[composite[ORDADD, cross[ordplus[x], Id]], composite[ordplus[x], ORDADD]] == True

In[52]:= composite[ordplus[x_], ORDADD] := composite[ORDADD, cross[ordplus[x], Id]]
```

Observation. The above rewrite rule implies that left-addition commutes with right-addition.

```
In[53]:= composite[ordplus[x], ORDADD, RIGHT[y]]

Out[53]= composite[ORDADD, RIGHT[y], ordplus[x]]
```

Theorem. A similar result for right-addition.

```
In[54]:= Assoc[composite[ORDADD, cross[Id, ORDADD]], ASSOC, RIGHT[x]] // Reverse

Out[54]= composite[ORDADD, RIGHT[x], ORDADD] ==
          composite[ORDADD, cross[Id, composite[ORDADD, RIGHT[x]]]]

In[55]:= composite[ORDADD, RIGHT[x_], ORDADD] :=
          composite[ORDADD, cross[Id, composite[ORDADD, RIGHT[x]]]]
```

Observation. The preceding rewrite rule rewrites the composite of two right-additions as a single right-addition.

```
In[56]:= composite[ORDADD, RIGHT[x], ORDADD, RIGHT[y]]

Out[56]= composite[ORDADD, RIGHT[ordadd[y, x]]]
```

special cases

Lemma. A simplification rule.

```
In[57]:= Assoc[id[OMEGA], id[range[ordplus[x]]], ordplus[x]]

Out[57]= composite[id[OMEGA], ordplus[x]] == ordplus[x]

In[58]:= composite[id[OMEGA], ordplus[x_]] := ordplus[x]
```

Lemma. A simplification rule.

```
In[59]:= Assoc[SUCC, id[OMEGA], ordplus[x]] // Reverse

Out[59]= composite[id[OMEGA], SUCC, ordplus[x]] == composite[SUCC, ordplus[x]]

In[60]:= composite[id[OMEGA], SUCC, ordplus[x_]] := composite[SUCC, ordplus[x]]
```

The following is a special case of the rule about left-addition commuting with right-addition. The orientation of this rule has been chosen so that it agrees with a similar rule for the addition of natural numbers.

Corollary. The function **ordplus[x]** commutes with **SUCC**.

```
In[61]:= SubstTest[composite, ordplus[x], ORDADD, RIGHT[y], y → set[0]] // Reverse

Out[61]= composite[ordplus[x], SUCC] == composite[SUCC, ordplus[x]]

In[62]:= composite[ordplus[x_], SUCC] := composite[SUCC, ordplus[x]]
```

other consequences of the associative law

A standard result about associative relations is the transitivity of their left and right divisibility relations. For left-divisibility, an explicit formula is available, so no new rule is needed.

```
In[63]:= leftdivisibility[ORDADD]

Out[63]= composite[id[OMEGA], S, id[OMEGA]]
```

For the case of right-divisibility the following holds.

Theorem. The right-divisibility relation for **ORDADD** is transitive.

```
In[64]:= SubstTest[implies, associative[t],
           TRANSITIVE[composite[t, inverse[SECOND]]], t → ORDADD] // Reverse

Out[64]= TRANSITIVE[composite[ORDADD, inverse[SECOND]]] == True

In[65]:= TRANSITIVE[composite[ORDADD, inverse[SECOND]]] := True
```