ordinal number wrapper

Johan G. F. Belinfante
2005 October 19

```
In[1]:= SetDirectory["l:"]; << goedel74.18a; << tools.m

        :Package Title: goedel74.18a            2005 October 18 at 2:00 p.m.

        It is now:  2005 Oct 19 at 8:34

        Loading Simplification Rules

        TOOLS.M                      Revised 2005 October 17

        weightlimit = 40
```

summary

An ordinal number wrapper is introduced. An application is presented to illustrate the use of this wrapper.

ord wrapper defined

Definition of **ord[x]**.

```
In[2]:= intersection[x_, image[V, intersection[OMEGA, set[x_]]]] := ord[x]
```

To prevent the wrapper from being introduced or eliminated accidentally, no membership rule will be introduced, although it would not be difficult to derive one.

ord wrapper basics

The following rule is needed when one wants to eliminate the **ord** wrapper in favor of a literal.

```
In[3]:= Map[assert,
         SubstTest[equal, x, intersection[x, image[V, intersection[y, set[x]]]], y → OMEGA]]

Out[3]= equal[x, ord[x]] == member[x, OMEGA]

In[4]:= equal[x_, ord[x_]] := member[x, OMEGA]
```

A conditional rewrite rule is useful when one wants to specialize results about ordinals to special types of ordinals, such as natural numbers or cardinals.

```
In[5]:= ord[x_] := x /; member[x, OMEGA]
```

The usefulness of the **ord** wrapper depends on the following:

```
In[6]:= SubstTest[member, intersection[x, image[V, intersection[y, set[x]]]], y, y → OMEGA]

Out[6]= member[ord[x], OMEGA] == True

In[7]:= member[ord[x_], OMEGA] := True
```

Corollary.

```
In[8]:= SubstTest[implies, member[u, v], member[u, V], {u → ord[x], v → OMEGA}]

Out[8]= member[ord[x], V] == True

In[9]:= member[ord[x_], V] := True
```

reify rule

It is useful to have a **reify** rule so that variable-free formulations are easy to derive.

```
In[10]:= SubstTest[reify, x, intersection[f[x], image[V, intersection[y, set[f[x]]]]], y → OMEGA]

Out[10]= reify[x, ord[f[x]]] ==
          composite[reify[x, f[x]], id[image[inverse[VERTSECT[reify[x, f[x]]], OMEGA]]]

In[11]:= reify[x_, ord[y_]] :=
          composite[reify[x, y], id[image[inverse[VERTSECT[reify[x, y]]], OMEGA]]]
```

an application

Lemma.

```
In[12]:= SubstTest[implies, member[y, OMEGA], subclass[y, OMEGA], y → ord[x]]

Out[12]= subclass[ord[x], OMEGA] == True

In[13]:= subclass[ord[x_], OMEGA] := True

In[14]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u → P[ord[x]], v → P[OMEGA], w -> union[fix[composite[E, BIGCAP]], set[0]]}]

Out[14]= subclass[P[ord[x]], union[fix[composite[E, BIGCAP]], set[0]]] == True

In[15]:= subclass[P[ord[x_]], union[fix[composite[E, BIGCAP]], set[0]]] := True
```

Corollary.

```
In[16]:= member[restrict[S, ord[x], ord[x]], WO]

Out[16]= True
```

Another application.

```
In[17]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
          {u → P[restrict[E, V, ord[x]]], v → P[restrict[E, V, OMEGA]], w → WF}]

Out[17]= WELLFOUNDED[composite[id[ord[x]], E]] == True

In[18]:= WELLFOUNDED[composite[id[ord[x_]], E]] := True
```

---

dichotomy

Lemma 1.

```
In[19]:= SubstTest[implies, and[member[u, OMEGA], member[v, OMEGA]],
          or[member[u, v], subclass[v, u]], {u → ord[x], v → ord[y]}]

Out[19]= or[member[ord[x], ord[y]], subclass[ord[y], ord[x]]] == True

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

Lemma 2.

```
In[21]:= SubstTest[implies, and[member[u, OMEGA], member[v, OMEGA]],
          not[and[member[u, v], subclass[v, u]]], {u → ord[x], v → ord[y]}]

Out[21]= or[not[member[ord[x], ord[y]]], not[subclass[ord[y], ord[x]]]] == True

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

Theorem.

```
In[23]:= equiv[subclass[ord[x], ord[y]], not[member[ord[y], ord[x]]]]

Out[23]= True

In[24]:= subclass[ord[x_], ord[y_]] := not[member[ord[y], ord[x]]]
```

Corollary

```
In[25]:= Map[not, SubstTest[subclass, ord[x], ord[y], y → x]] // Reverse

Out[25]= member[ord[x], ord[x]] == False

In[26]:= member[ord[x_], ord[x_]] := False
```

Lemma.

```
In[27]:= SubstTest[implies, member[y, OMEGA], member[U[y], OMEGA], y → ord[x]]

Out[27]= member[U[ord[x]], OMEGA] == True

In[28]:= member[U[ord[x_]], OMEGA] := True
```

Lemma.

```
In[29]:= SubstTest[subclass, ord[z], ord[y], z → U[ord[x]]]

Out[29]= subclass[U[ord[x]], ord[y]] == not[member[ord[y], U[ord[x]]]]

In[30]:= subclass[U[ord[x_]], ord[y_]] := not[member[ord[y], U[ord[x]]]]
```

Any ordinal is full.

```
In[31]:= Map[not, SubstTest[implies, member[y, OMEGA], full[y], y → ord[x]]]

Out[31]= member[ord[x], U[ord[x]]] == False

In[32]:= member[ord[x_], U[ord[x_]]] := False
```

---

complete lattice example

A general result.

```
In[33]:= Map[member[0, domain[#]] &, GLB[composite[id[x], S, id[x]]] // RelnNormality]

Out[33]= member[0, domain[GLB[composite[id[x], S, id[x]]]]] == member[U[x], x]

In[34]:= member[0, domain[GLB[composite[id[x_], S, id[x_]]]]] := member[U[x], x]
```

Lemma.

```
In[35]:= SubstTest[implies, and[member[y, WO], member[0, domain[GLB[y]]]],
          member[y, CL], y → restrict[S, ord[x], ord[x]]]

Out[35]= or[member[composite[id[ord[x]], S, id[ord[x]]], CL],
           not[member[U[ord[x]], ord[x]]]] == True

In[36]:= (% /. x → x_) /. Equal → SetDelayed
```

Conversely:

```
In[37]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
            {p1 → member[y, CL], p2 -> equal[P[fix[y]], domain[GLB[y]]],
             p3 → member[0, domain[GLB[y]]]}]] /. y -> composite[id[ord[x]], S, id[ord[x]]]

Out[37]= or[member[U[ord[x]], ord[x]],
           not[member[composite[id[ord[x]], S, id[ord[x]]], CL]]] == True

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

These results can be combined into one rewrite rule:

```
In[39]:= equiv[member[composite[id[ord[x]], S, id[ord[x]]], CL], member[U[ord[x]], ord[x]]]
Out[39]= True

In[40]:= member[composite[id[ord[x_]], S, id[ord[x_]]], CL] := member[U[ord[x]], ord[x]]
```

Corollary.

```
In[41]:= SubstTest[member, composite[id[ord[y]], S, id[ord[y]]], CL, y → succ[ord[x]]]
Out[41]= member[composite[id[succ[ord[x]]], S, id[succ[ord[x]]]], CL] == True

In[42]:= member[composite[id[succ[ord[x_]]], S, id[succ[ord[x_]]]], CL] := True
```

The wrapper can be removed:

```
In[43]:= SubstTest[implies, equal[y, ord[x]],
          member[composite[id[succ[y]], S, id[succ[y]]], CL], y → x]
Out[43]= or[member[composite[id[succ[x]], S, id[succ[x]]], CL], not[member[x, OMEGA]]] == True

In[44]:= or[member[composite[id[succ[x_]], S, id[succ[x_]]], CL], not[member[x_, OMEGA]]] := True
```

---

a counterexample

The natural numbers do not form a complete lattice with respect to inclusion because there is no greatest number.

```
In[45]:= SubstTest[member, composite[id[ord[x]], S, id[ord[x]]], CL, x → omega]
Out[45]= member[composite[id[omega], S, id[omega]], CL] == False

In[46]:= member[composite[id[omega], S, id[omega]], CL] := False
```

Adding a top element in this case yields a complete lattice.

```
In[47]:= SubstTest[member, composite[id[ord[x]], S, id[ord[x]]], CL, x → succ[omega]]
Out[47]= member[composite[id[succ[omega]], S, id[succ[omega]]], CL] == True

In[48]:= member[composite[id[succ[omega]], S, id[succ[omega]]], CL] := True
```

Lemma. (The successor of **omega** is wellordered by inclusion.)

```
In[49]:= SubstTest[subclass, P[ord[x]],
          union[fix[composite[E, BIGCAP]], set[0]], x → succ[omega]]
Out[49]= subclass[P[succ[omega]], union[fix[composite[E, BIGCAP]], set[0]]] == True

In[50]:= subclass[P[succ[omega]], union[fix[composite[E, BIGCAP]], set[0]]] := True
```

Counterexample: a wellordered complete lattice need not be finite.

```
In[51]:= Map[not, SubstTest[implies, and[member[x, y], subclass[y, z]],
           member[x, z], {x → composite[id[succ[omega]], S, id[succ[omega]]],
            y → intersection[WO, CL], z → FINITE}]]

Out[51]= subclass[intersection[CL, WO], FINITE] == False

In[52]:= subclass[intersection[CL, WO], FINITE] := False
```

Corollary 1.

```
In[53]:= Map[not, SubstTest[implies, subclass[u, w],
           subclass[intersection[u, v], w], {u → CL, v → WO, w → FINITE}]]

Out[53]= subclass[CL, FINITE] == False

In[54]:= subclass[CL, FINITE] := False
```

Corollary 2.

```
In[55]:= Map[not, SubstTest[implies, subclass[v, w],
           subclass[intersection[u, v], w], {u → CL, v → WO, w → FINITE}]]

Out[55]= subclass[WO, FINITE] == False

In[56]:= subclass[WO, FINITE] := False
```

other counterexamples

Other counterexamples involving finiteness are presented here.

```
In[57]:= SubstTest[member, composite[Id, x], FINITE, x → composite[id[omega], E]]

Out[57]= member[composite[id[omega], E], FINITE] == False

In[58]:= member[composite[id[omega], E], FINITE] := False

In[59]:= Map[not, SubstTest[implies, and[member[x, y], subclass[y, z]],
           member[x, z], {x → restrict[E, omega, omega], y -> WF, z -> FINITE}]]

Out[59]= subclass[WF, FINITE] == False

In[60]:= subclass[WF, FINITE] := False
```