# a rewrite rule for U[X[x]]

*Johan G. F. Belinfante*
*2004 October 31*

```
In[1]:= SetDirectory["i:"]; << goedel62.30b; << tools.m

        :Package Title: goedel62.30b         2004 October 30 at 9:15 p.m.

        It is now:  2004 Oct 31 at 10:16

        Loading Simplification Rules

        TOOLS.M                       Revised 2004 October 28

        weightlimit = 40
```

## summary

An explicit formula for **U[X[x]]** is derived that does not depend on whether or not the axiom of choice holds. The idea is this: Suppose that there exists a cross-section **w** for **x**. If **pair[u,v]** is any point of **composite[Id, x]**, then one can construct another cross-section **z** of **x** which goes through that point by simply replacing the value that the function **w** assigns to the point **u** in **domain[w] = domain[x]** with the new value **v**. Hence, either **X[x]=0** or else **U[X[x]] = composite[Id,x]**. These two cases can be combined into a single formula by including a factor **id[image[V, X[x]]]**. The equation will be established by showing that each side is contained in the other.

## the easy direction

Lemma.

```
In[2]:= SubstTest[subclass, intersection[u, v], u, {u → FUNS,
          v → intersection[P[x], image[inverse[IMAGE[FIRST]], singleton[domain[x]]]]}]

Out[2]= subclass[X[x], FUNS] == True

In[3]:= subclass[X[x_], FUNS] := True
```

Corollary.

```
In[4]:= equal[intersection[FUNS, X[x]], X[x]]
Out[4]= True
In[5]:= intersection[FUNS, X[x_]] := X[x]
```

Lemma.

```
In[6]:= SubstTest[implies, subclass[u, v], subclass[U[u], U[v]], {u -> X[x], v → FUNS}]
Out[6]= subclass[U[X[x]], cart[V, V]] == True
In[7]:= subclass[U[X[x_]], cart[V, V]] := True
```

---

## the harder direction

Lemma. Exchanging one point for another.

```
In[8]:= FUNCTION[union[composite[funpart[w], id[complement[singleton[u]]]],
          cart[singleton[u], singleton[v]]]] // AssertTest
Out[8]= FUNCTION[union[cart[singleton[u], singleton[v]],
          composite[funpart[w], id[complement[singleton[u]]]]]] == True
In[9]:= (% /. {u → u_, v → v_, w → w_}) /. Equal → SetDelayed
```

Lemma concerning domains.

```
In[10]:= implies[and[member[u, domain[funpart[w]]], member[v, V]],
           equal[domain[funpart[w]],
            domain[union[composite[funpart[w], id[complement[singleton[u]]]],
              cart[singleton[u], singleton[v]]]]] // AssertTest
Out[10]= or[equal[domain[funpart[w]],
            union[intersection[complement[singleton[u]], domain[funpart[w]]],
             intersection[image[V, singleton[v]], singleton[u]]]],
           not[member[u, domain[funpart[w]]]], not[member[v, V]]] == True
In[11]:= (% /. {u → u_, v → v_, w → w_}) /. Equal → SetDelayed
```

Lemma concerning sethood of **z**.

```
In[12]:= SubstTest[or, member[z, V],
          not[equal[z, union[singleton[s], intersection[w, t]]]],
          not[member[w, V]], {s → PAIR[u, v], t → cart[x, V]}]

Out[12]= or[member[z, V],
           not[equal[z, union[cart[singleton[u], singleton[v]], composite[w, id[x]]]]],
           not[member[w, V]]] == True

In[13]:= (% /. {u → u_, v → v_, w → w_, x → x_, z → z_}) /. Equal → SetDelayed
```

Lemma concerning membership in the sum class of **X[x]**.

```
In[14]:= SubstTest[implies, and[member[t, z], member[z, y]], member[t, U[y]], y → X[x]]

Out[14]= or[member[t, U[X[x]]], not[equal[domain[x], domain[z]]], not[FUNCTION[z]],
           not[member[t, z]], not[member[z, V]], not[subclass[z, x]]] == True

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

Lemma about the domain of the new cross-section **z**.

```
In[16]:= Map[not, SubstTest[and, implies[and[p1, p2, p3], p6],
           implies[p2, p7], implies[and[p3, p6, p7], p8],
           implies[and[p1, p8], p9], not[implies[and[p1, p2, p3], p9]],
           {p1 → equal[domain[x], domain[funpart[w]]],
            p2 → member[pair[u, v], composite[Id, x]],
            p3 → equal[z, union[composite[funpart[w], id[complement[singleton[u]]]],
               cart[singleton[u], singleton[v]]]],
            p6 → member[u, domain[funpart[w]]], p7 → member[v, V], p8 ->
             equal[domain[z], domain[funpart[w]]], p9 → equal[domain[z], domain[x]]}]]

Out[16]= or[equal[domain[x], domain[z]],
           not[equal[z, union[cart[singleton[u], singleton[v]],
              composite[funpart[w], id[complement[singleton[u]]]]]]],
           not[equal[domain[x], domain[funpart[w]]]], not[member[u, V]],
           not[member[v, V]], not[member[pair[u, v], x]]] == True

In[17]:= (% /. {u → u_, v → v_, w → w_, x → x_, z → z_}) /. Equal → SetDelayed
```

The main argument, but not yet cleaned up. This takes a while.

```
In[18]:= Map[not, SubstTest[and, implies[p1, p4],
           implies[p1, p5], implies[p1, p9], implies[p1, p10],
           implies[and[p1, p4, p5, p9, p10], p11], implies[p1, p12],
           implies[and[p11, p12], p13], not[implies[p1, p13]],
           {p1 → and[member[funpart[w], X[x]], member[pair[u, v], composite[Id, x]],
              equal[z, union[composite[funpart[w], id[complement[singleton[u]]]],
                cart[singleton[u], singleton[v]]]]], p4 → FUNCTION[z],
            p5 → subclass[z, x], p6 → member[u, domain[funpart[w]]],
            p7 → member[v, V], p8 -> equal[domain[z], domain[funpart[w]]],
            p9 → equal[domain[z], domain[x]], p10 → member[z, V], p11 → member[z, X[x]],
            p12 → member[pair[u, v], z], p13 → member[pair[u, v], U[X[x]]]}]] /.
         z -> union[composite[funpart[w], id[complement[singleton[u]]]],
           cart[singleton[u], singleton[v]]]

Out[18]= or[member[pair[u, v], U[X[x]]], not[equal[domain[x], domain[funpart[w]]]],
           not[member[u, V]], not[member[v, V]], not[member[funpart[w], V]],
           not[member[pair[u, v], x]], not[subclass[funpart[w], x]]] == True

In[19]:= (% /. {u → u_, v → v_, w → w_, x → x_}) /. Equal → SetDelayed
```

---

## eliminating the variables

Elimination of the variables **u** and **v**.

```
In[20]:= Map[equal[0, composite[Id, complement[#]]] &, SubstTest[class, pair[u, v],
           implies[and[member[s, t], member[pair[u, v], composite[Id, x]]],
            member[pair[u, v], y]], {s → funpart[w], t → X[x], y → U[X[x]]}]] // Reverse

Out[20]= or[not[equal[domain[x], domain[funpart[w]]]], not[member[funpart[w], V]],
           not[subclass[funpart[w], x]], subclass[composite[Id, x], U[X[x]]]] == True

In[21]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Removal of the variable **w**.

```
In[22]:= Map[equal[V, #] &,
           SubstTest[class, w, implies[member[funpart[w], t], subclass[u, v]],
            {t → X[x], u → composite[Id, x], v → U[X[x]]}]] // Reverse

Out[22]= or[equal[0, X[x]], subclass[composite[Id, x], U[X[x]]]] == True

In[23]:= (% /. x → x_) /. Equal → SetDelayed
```

The use of **AssertTest** suffices to combine the two directions, and to combine the case that **X[x]** is empty with the case that it is not.

```
In[24]:= equal[U[X[x]], composite[x, id[image[V, X[x]]]]] // AssertTest
Out[24]= equal[composite[x, id[image[V, X[x]]]], U[X[x]]] == True
```

The final result is a simple rewrite rule for the sum class of the class of cross-sections of **x**.

```
In[25]:= U[X[x_]] := composite[x, id[image[V, X[x]]]]
```

---

## corollary

As a corollary, one obtains the following rule for the class of cross-sections to be a set.

```
In[27]:= SubstTest[member, U[y], V, y → X[x]] // Reverse
Out[27]= member[X[x], V] == or[equal[0, X[x]], member[range[x], V]]

In[28]:= member[X[x_], V] := or[equal[0, X[x]], member[range[x], V]]
```