wo => axch

Johan G. F. Belinfante
2005 July 20

```
In[1]:= SetDirectory["i:"]; << goedel71.18b; << tools.m
```

```
:Package Title: goedel71.18b          2005 July 18 at 11:35 p.m.

It is now:  2005 Jul 20 at 1:49

Loading Simplification Rules

TOOLS.M                       Revised 2005 July 18

weightlimit = 40
```

---

summary

The cardinality of a set **x** is by definition the intersection of the class of all ordinals that are equipollent to the set:

```
In[2]:= A[intersection[OMEGA, image[Q, set[x]]]]
```

```
Out[2]= card[x]
```

Here **OMEGA** is the class of all ordinals, and **Q** is the equipollence relation:

```
In[3]:= class[pair[x, y],
         exists[z, and[ONEONE[z], equal[x, domain[z]], equal[y, range[z]]]]]
```

```
Out[3]= Q
```

Since any nonempty class of ordinals has a least member, it follows that **card[x]** is an ordinal, unless there does not exist any ordinal equipollent to the set **x**. In the latter case **card[x] = A[0] = V** is the universal class, which is not a set.

```
In[4]:= member[card[x], OMEGA]
```

```
Out[4]= member[x, image[Q, OMEGA]]
```

The class **image[Q, OMEGA]** is the class of all sets which are equipollent to some ordinal. In this notebook it is shown that if every set is equipollent to an ordinal, then the axiom of choice holds. (The converse also holds, but is not addressed here.)

preliminaries and the strategy of the derivation

The axiom of choice is equivalent to the statement that every set has a cross-section, that is, a function contained in x with the same domain as **x**. If the set **x** holds no ordered pairs, then its domain is empty, and the empty set is the sole cross-section. The class of all cross-sections of **x** is denoted by **X[x]**, and the class of all sets that admit cross-sections is

```
In[5]:= class[x, not[empty[X[x]]]]
Out[5]= SELECT
```

Thus the statement that every set admits a cross-section is:

```
In[6]:= equal[V, SELECT]
Out[6]= axch
```

The derivation makes use of a basic theorem that is useful for proving various equivalents of the axiom of choice. It says that the class **SELECT** is closed under compatible compositions:

```
In[7]:= implies[and[member[u, SELECT], member[v, SELECT],
          subclass[range[v], domain[u]]], member[composite[u, v], SELECT]]
Out[7]= True
```

This theorem will be applied in the next section to the case where **u** is a one-to-one correspondence between an ordinal and the range of **x**. A variable **w** will be introduced to represent the one-to-one correspondence, and the condition that it be one-to-one is introduced by wrapping it with **oopart**. Later this wrapper is removed by using the following fact:

```
In[8]:= equal[w, oopart[w]]
Out[8]= and[FUNCTION[w], FUNCTION[inverse[w]]]
```

After deriving the existence of a cross-section under these hypotheses, the variables are removed to obtain a variable-free formulation.

derivation

The main idea is to use the invariance of **SELECT** under compatible composition, as explained above.

```
In[9]:= Map[or[not[equal[0, X[composite[id[range[oopart[w]]], x]]]], #] &,
         SubstTest[implies, and[member[u, SELECT], member[v, SELECT],
           subclass[range[v], domain[u]]], member[composite[u, v], SELECT],
          {u → oopart[w], v → composite[inverse[oopart[w]], x]}]]

Out[9]= or[equal[0, X[composite[inverse[oopart[w]], x]]],
          not[equal[0, X[composite[id[range[oopart[w]]], x]]]],
          not[member[composite[inverse[oopart[w]], x], V]],
          not[member[oopart[w], V]]] == True

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

The result can be cleaned up. A series of lemmas will be used to do this.

```
In[11]:= SubstTest[or, member[composite[z, x], V],
          not[FUNCTION[z]], not[member[x, V]], z → inverse[oopart[w]]]

Out[11]= or[member[composite[inverse[oopart[w]], x], V], not[member[x, V]]] == True

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

Lemma. (Any relation whose range is contained in **OMEGA** has a cross-section.)

```
In[13]:= SubstTest[implies, and[member[z, V], member[range[z], P[OMEGA]]],
          not[equal[0, X[z]]], z -> composite[inverse[oopart[w]], x]]

Out[13]= or[not[equal[0, X[composite[inverse[oopart[w]], x]]]],
           not[member[composite[inverse[oopart[w]], x], V]],
           not[subclass[image[inverse[oopart[w]], range[x]], OMEGA]]] == True

In[14]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[15]:= Map[not, SubstTest[and, implies[p1, p2], implies[p3, p4],
           implies[p4, p5], implies[and[p2, p5], p6], not[implies[and[p1, p3], p6]],
           {p1 → member[x, V], p2 -> member[composite[inverse[oopart[w]], x], V],
            p3 → member[domain[oopart[w]], OMEGA],
            p4 → subclass[domain[oopart[w]], OMEGA],
            p5 -> subclass[image[inverse[oopart[w]], range[x]], OMEGA],
            p6 -> not[equal[0, X[composite[inverse[oopart[w]], x]]]]}]]

Out[15]= or[not[equal[0, X[composite[inverse[oopart[w]], x]]]],
           not[member[x, V]], not[member[domain[oopart[w]], OMEGA]]] == True

In[16]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

The main theorem is obtained by combining the various lemmas.

```
In[17]:= Map[not, SubstTest[and, implies[p2, p5], implies[p3, p6],
           implies[and[p6, p7], p8], implies[and[p1, p5, p8], p9],
           implies[and[p2, p4], not[p9]], not[implies[and[p1, p2, p3, p4], not[p7]]],
           {p1 -> member[oopart[w], V], p2 → member[x, V], p3 →
             equal[range[oopart[w]], range[x]], p4 → member[domain[oopart[w]], OMEGA],
            p5 -> member[composite[inverse[oopart[w]], x], V], p6 →
             equal[X[x], X[composite[id[range[oopart[w]]], x]]], p7 → equal[0, X[x]],
            p8 -> equal[0, X[composite[id[range[oopart[w]]], x]]],
            p9 -> equal[0, X[composite[inverse[oopart[w]], x]]]}]]

Out[17]= or[not[equal[0, X[x]]],
           not[equal[range[x], range[oopart[w]]]], not[member[x, V]],
           not[member[domain[oopart[w]], OMEGA]], not[member[oopart[w], V]]] == True

In[18]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

The **oopart** wrapper has served its purpose and is now removed.

```
In[19]:= SubstTest[implies, equal[v, oopart[w]],
          or[not[equal[0, X[x]]], not[equal[range[x], range[v]]], not[member[x, V]],
           not[member[domain[v], OMEGA]], not[member[v, V]]], v → w]

Out[19]= or[not[equal[0, X[x]]], not[equal[range[w], range[x]]],
           not[FUNCTION[w]], not[FUNCTION[inverse[w]]], not[member[w, V]],
           not[member[x, V]], not[member[domain[w], OMEGA]]] == True

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

This can be restated more succinctly using the classes **BIJ** and **SELECT**.

```
In[21]:= implies[and[member[x, V], member[w, BIJ],
           member[domain[w], OMEGA], equal[range[w], range[x]]], member[x, SELECT]]

Out[21]= True
```

The next step is to use Gödel's algorithm in the form of rewrite rules for **class** to remove the variables. The more complicated concepts are sequestered from the action of Gödel's algorithm by using dummy variables. For example, the one-to-one hypothesis is hidden from the action of the **class** rules by introducing a temporary dummy variable **t** later set equal to the class **BIJ** of all bijections.

```
In[22]:= Map[equal[0, composite[Id, complement[#]]] &,
           SubstTest[class, pair[x, w], implies[and[member[x, V], member[w, t],
              member[domain[w], u], equal[range[w], range[x]]], member[x, s]],
            {s → SELECT, t → BIJ, u → OMEGA}]] // Reverse

Out[22]= equal[0, intersection[OMEGA,
            image[Q, image[IMAGE[SECOND], complement[SELECT]]]]] == True

In[23]:= intersection[OMEGA, image[Q, image[IMAGE[SECOND], complement[SELECT]]]] := 0
```

A rewrite rule about disjoint images is invoked to transform this statement:

```
In[24]:= SubstTest[disjoint, y, image[inverse[z], x],
          {x → OMEGA, y → image[IMAGE[SECOND], complement[SELECT]], z → Q}]

Out[24]= subclass[image[inverse[IMAGE[SECOND]], image[Q, OMEGA]], SELECT] == True

In[25]:= subclass[image[inverse[IMAGE[SECOND]], image[Q, OMEGA]], SELECT] := True
```

The last step uses equality substitution, and **assert** is used for simplification.

```
In[26]:= Map[assert, SubstTest[implies,
           and[equal[u, v], subclass[image[x, u], y]], subclass[image[x, v], y],
           {u → image[Q, OMEGA], v → V, x -> inverse[IMAGE[SECOND]], y → SELECT}]]

Out[26]= or[axch, not[equal[V, image[Q, OMEGA]]]] == True

In[27]:= or[axch, not[equal[V, image[Q, OMEGA]]]] := True
```