a8 => axch

Johan G. F. Belinfante
2004 November 23

```
In[1]:= SetDirectory["i:"]; << goedel63.21a; << tools.m

        :Package Title: goedel63.21a        2004 November 21 at 11:25 p.m.

        It is now:  2004 Nov 23 at 16:26

        Loading Simplification Rules

        TOOLS.M                       Revised 2004 November 17

        weightlimit = 40
```

summary

This notebook is devoted to the proof that Bertrand Russell's form **a8** of the axiom of choice implies the cross-section form **ac2**. The easier reverse implication had already been established previously. Recall that Russell's form of the axiom of choice says that for any pairwise disjoint collection **x** of nonempty sets, there is a set **z** whose intersection with each member of **x** is a singleton. The form **ac2** says that any set **x** has a subset **y** which is a function with the same domain. The idea of the proof is to apply Russell's statement to the pairwise disjoint collection of nonempty vertical strips obtained from **x** by making vertical sections. Each vertical strip in this collection is a cartesian product **cart[singleton[u], image[x, singleton[u]]]**, where **u** is an element of the domain of **x**. The main obstacle that had to be overcome was to verify how the selecting set **z** yields a cross-section of **x**.

a temporary abbreviation

For convenience, the following abbreviation will be adopted in this notebook:

```
In[2]:= a8 := subclass[
          intersection[cliques[union[DISJOINT, Id]], P[complement[singleton[0]]]],
          domain[UB[image[inverse[CAP], range[SINGLETON]]]]]
```

Note that **cliques[union[DISJOINT,Id]]** is the class of all pairwise disjoint collections of sets, **P[complement[singleton[0]]]** is the class of all collections of non-empty sets, and the class **domain[UB[image[inverse[CAP],range[SINGLETON]]]]** is the class of all sets for which there is a set whose intersection with each member is a singleton.

---

axch => a8

The implication **axch => a8** has already been established earlier, and is available in the **GOEDEL** program as a temporary rewrite rule:

```
In[3]:= or[not[axch], subclass[
          intersection[cliques[union[DISJOINT, Id]], P[complement[singleton[0]]]],
          domain[UB[image[inverse[CAP], range[SINGLETON]]]]]]

Out[3]= True
```

Restatement:

```
In[4]:= implies[axch, a8]

Out[4]= True
```

The results of this notebook will justify replacing this temporary rule with the better rule that says in effect **a8 := axch**.

---

a pairwise disjoint collection

If **x** is a function, then the class **image[CART, composite[x, inverse[SINGLETON]]]** is a pairwise disjoint collection of sets. The first order of business is to show that this class is a set when **x** is a set. To show this, one needs this function:

```
In[5]:= abstract[x, image[CART, composite[x, inverse[SINGLETON]]]]

Out[5]= composite[CART, cross[SINGLETON, Id]]
```

The axiom of replacement is applied to this function:

```
In[6]:= SubstTest[implies, and[member[y, V], FUNCTION[z]],
         member[image[z, y], V], {z -> composite[CART, cross[SINGLETON, Id]],
          y -> composite[VERTSECT[setpart[x]], id[domain[setpart[x]]]]}]

Out[6]= member[image[CART, composite[VERTSECT[setpart[x]],
            id[domain[setpart[x]]], inverse[SINGLETON]]], V] == True

In[7]:= (% /. x → x_) /. Equal → SetDelayed
```

The pairwise disjointness statement is:

```
In[8]:= SubstTest[implies, FUNCTION[w],
         subclass[cart[image[CART, composite[w, inverse[SINGLETON]]],
           image[CART, composite[w, inverse[SINGLETON]]]], union[DISJOINT, Id]],
         w -> composite[VERTSECT[setpart[x]], id[domain[setpart[x]]]]]

Out[8]= subclass[
          cart[image[CART, composite[VERTSECT[setpart[x]], id[domain[setpart[x]]],
             inverse[SINGLETON]]], image[CART, composite[VERTSECT[setpart[x]],
             id[domain[setpart[x]]], inverse[SINGLETON]]]], union[DISJOINT, Id]] == True

In[9]:= (% /. x → x_) /. Equal → SetDelayed
```

Russell's statement is applied to this pairwise disjoint collection of vertical strips:

```
In[10]:= SubstTest[implies, and[member[u, v], subclass[v, w]], member[u, w],
          {u -> image[CART, composite[VERTSECT[setpart[x]],
              id[domain[setpart[x]]], inverse[SINGLETON]]], v ->
            intersection[cliques[union[DISJOINT, Id]], P[complement[singleton[0]]]],
           w -> domain[UB[image[inverse[CAP], range[SINGLETON]]]]}]

Out[10]= or[not[
            equal[0, ub[image[inverse[CAP], range[SINGLETON]], image[CART, composite[
                VERTSECT[setpart[x]], id[domain[setpart[x]]], inverse[SINGLETON]]]]]],
           not[subclass[intersection[cliques[union[DISJOINT, Id]],
              P[complement[singleton[0]]]],
             domain[UB[image[inverse[CAP], range[SINGLETON]]]]]]] == True

In[11]:= (% /. x → x_) /. Equal → SetDelayed
```

---

extra IMAGE factor

Functions can be characterized by the fact that their vertical sections are singletons. A rewrite rule to this effect was derived earlier, but here this result is needed with an extra **IMAGE** factor. The new result can be deduced from the old one using this rewrite rule:

```
In[12]:= composite[IMAGE[z], VERTSECT[x]] // RelnNormality // Reverse

Out[12]= composite[VERTSECT[composite[z, x]], id[domain[VERTSECT[x]]]] ==
          composite[IMAGE[z], VERTSECT[x]]

In[13]:= composite[VERTSECT[composite[z_, x_]], id[domain[VERTSECT[x_]]]] :=
          composite[IMAGE[z], VERTSECT[x]]
```

The result is:

```
In[14]:= Map[subclass[#, range[SINGLETON]] &,
          ImageComp[VERTSECT[composite[z, x]], id[domain[VERTSECT[x]]], y]]

Out[14]= subclass[image[IMAGE[z], image[VERTSECT[x], y]], range[SINGLETON]] ==
          and[FUNCTION[composite[thinpart[composite[z, x]],
             id[intersection[y, domain[VERTSECT[x]]]]], subclass[
            intersection[y, domain[VERTSECT[x]]], image[inverse[x], domain[z]]]]

In[15]:= subclass[image[IMAGE[z_], image[VERTSECT[x_], y_]], range[SINGLETON]] :=
          and[FUNCTION[composite[thinpart[composite[z, x]],
             id[intersection[y, domain[VERTSECT[x]]]]], subclass[
            intersection[y, domain[VERTSECT[x]]], image[inverse[x], domain[z]]]]
```

---

cross identity

The following **cross** identity was established earlier for the special case that **z = V**, but in this notebook, this more general result is needed.

```
In[16]:= composite[cross[x, y], id[id[z]]] // RelnNormality

Out[16]= composite[cross[x, y], id[id[z]]] ==
          composite[intersection[composite[inverse[FIRST], x],
            composite[inverse[SECOND], y]], id[z], inverse[DUP]]

In[17]:= composite[cross[x_, y_], id[id[z_]]] :=
          composite[intersection[composite[inverse[FIRST], x],
            composite[inverse[SECOND], y]], id[z], inverse[DUP]]
```

Application:

```
In[18]:= (SubstTest[subclass,
            image[IMAGE[id[z]], image[VERTSECT[cross[Id, w]], id[domain[w]]]],
            range[SINGLETON], w → setpart[x]]) // InvertFix

Out[18]= subclass[image[IMAGE[id[z]], image[CART, composite[VERTSECT[setpart[x]],
             id[domain[setpart[x]]], inverse[SINGLETON]]]], range[SINGLETON]] ==
          and[FUNCTION[composite[Id, intersection[z, setpart[x]]]],
           subclass[domain[setpart[x]], fix[composite[inverse[z], setpart[x]]]]]

In[19]:= subclass[image[IMAGE[id[z_]], image[CART, composite[VERTSECT[setpart[x_]],
             id[domain[setpart[x_]]], inverse[SINGLETON]]]], range[SINGLETON]] :=
          and[FUNCTION[composite[Id, intersection[z, setpart[x]]]],
           subclass[domain[setpart[x]], fix[composite[inverse[z], setpart[x]]]]]
```

---

hiding the construction

The idea now is to hide the details of the construction. The following lemma eliminates the variable **z** from the conclusion:

```
In[20]:= SubstTest[implies, member[u, v], not[empty[v]],
          {u -> composite[Id, intersection[setpart[x], z]], v -> X[setpart[x]]}]

Out[20]= or[not[equal[0, X[setpart[x]]]],
           not[equal[domain[setpart[x]], fix[composite[inverse[z], setpart[x]]]]],
           not[FUNCTION[composite[Id, intersection[z, setpart[x]]]]] == True

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

An application of **AssertTest** is needed to convert an inclusion to an equivalent equation, needed for pattern matching.

```
In[22]:= implies[subclass[domain[setpart[x]], fix[composite[inverse[z], setpart[x]]]],
           equal[domain[setpart[x]],
            fix[composite[inverse[z], setpart[x]]]]] // AssertTest

Out[22]= or[equal[domain[setpart[x]], fix[composite[inverse[z], setpart[x]]]],
           not[subclass[domain[setpart[x]],
             fix[composite[inverse[z], setpart[x]]]]] == True

In[23]:= (% /. {x → x_, z → z_}) /. Equal → SetDelayed
```

An application of **NotNotTest** is needed to simplify the membership statements in the following.

```
In[24]:= implies[
           member[z, ub[image[inverse[CAP], range[SINGLETON]], image[CART, composite[
               VERTSECT[setpart[x]], id[domain[setpart[x]]], inverse[SINGLETON]]]]],
           member[composite[Id, intersection[setpart[x], z]],
            X[setpart[x]]]] // NotNotTest

Out[24]= or[and[not[equal[0, domain[setpart[x]]]], not[
             subclass[domain[setpart[x]], fix[composite[inverse[z], setpart[x]]]]]],
           equal[domain[setpart[x]], fix[composite[inverse[z], setpart[x]]]],
           not[FUNCTION[composite[Id, intersection[z, setpart[x]]]]],
           not[member[z, V]]] == True

In[25]:= (% /. {x → x_, z → z_}) /. Equal → SetDelayed
```

The construction is now hidden:

```
In[26]:= Map[not,
          SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]], {p1 ->
             member[z, ub[image[inverse[CAP], range[SINGLETON]], image[CART, composite[
                 VERTSECT[setpart[x]], id[domain[setpart[x]]], inverse[SINGLETON]]]]],
            p2 -> member[composite[Id, intersection[setpart[x], z]], X[setpart[x]]],
            p3 → not[equal[0, X[setpart[x]]]]}]]

Out[26]= or[not[equal[0, X[setpart[x]]]],
           not[FUNCTION[composite[Id, intersection[z, setpart[x]]]]],
           not[member[z, V]], not[subclass[domain[setpart[x]],
             fix[composite[inverse[z], setpart[x]]]]] == True

In[27]:= (% /. {x → x_, z → z_}) /. Equal → SetDelayed
```

Another application of **NotNotTest** is needed to prepare for the next step.

```
In[28]:= implies[
           member[z, ub[image[inverse[CAP], range[SINGLETON]], image[CART, composite[
               VERTSECT[setpart[x]], id[domain[setpart[x]]], inverse[SINGLETON]]]]],
           not[empty[X[setpart[x]]]]] // NotNotTest

Out[28]= or[and[not[equal[0, domain[setpart[x]]]], not[
             subclass[domain[setpart[x]], fix[composite[inverse[z], setpart[x]]]]]],
           not[equal[0, X[setpart[x]]]],
           not[FUNCTION[composite[Id, intersection[z, setpart[x]]]]],
           not[member[z, V]]] == True

In[29]:= (% /. {x → x_, z → z_}) /. Equal → SetDelayed
```

The variable **z** is now elinminated:

```
In[30]:= Map[equal[V, #] &, SubstTest[class, z, implies[member[z, u], not[empty[v]]],
           {u -> ub[image[inverse[CAP], range[SINGLETON]],
              image[CART, composite[VERTSECT[setpart[x]], id[domain[setpart[x]]],
                inverse[SINGLETON]]]], v → X[setpart[x]]}]] // Reverse

Out[30]= or[equal[0, ub[image[inverse[CAP], range[SINGLETON]],
            image[CART, composite[VERTSECT[setpart[x]], id[domain[setpart[x]]],
              inverse[SINGLETON]]]]], not[equal[0, X[setpart[x]]]]] == True

In[31]:= (% /. x → x_) /. Equal → SetDelayed
```

The next step is to eliminate the literal containing **ub**.

```
In[32]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 → a8, p2 -> not[empty[ub[image[inverse[CAP], range[SINGLETON]],
                image[CART, composite[VERTSECT[setpart[x]], id[domain[setpart[x]]],
                  inverse[SINGLETON]]]]]], p3 → member[setpart[x], SELECT]}]]

Out[32]= or[not[equal[0, X[setpart[x]]]], not[subclass[
            intersection[cliques[union[DISJOINT, Id]], P[complement[singleton[0]]]],
            domain[UB[image[inverse[CAP], range[SINGLETON]]]]]]] == True

In[33]:= (% /. x → x_) /. Equal → SetDelayed
```

---

eliminating the variable x

Eliminating the variable **x** yields the result that **a8 => axch**.

```
In[34]:= Map[equal[V, #] &,
           SubstTest[class, x, implies[subclass[u, v], member[setpart[x], w]], {u →
              intersection[cliques[union[DISJOINT, Id]], P[complement[singleton[0]]]],
            v -> domain[UB[image[inverse[CAP], range[SINGLETON]]]],
            w → SELECT}]] // Reverse

Out[34]= or[axch, not[subclass[
            intersection[cliques[union[DISJOINT, Id]], P[complement[singleton[0]]]],
            domain[UB[image[inverse[CAP], range[SINGLETON]]]]]]] == True

In[35]:= % /. Equal → SetDelayed
```

It follows that **a8** and **axch** are equivalent.

```
In[36]:= equiv[subclass[
           intersection[cliques[union[DISJOINT, Id]], P[complement[singleton[0]]]],
           domain[UB[image[inverse[CAP], range[SINGLETON]]]]], axch]

Out[36]= True
```

```
In[37]:= subclass[
           intersection[cliques[union[DISJOINT, Id]], P[complement[singleton[0]]]],
           domain[UB[image[inverse[CAP], range[SINGLETON]]]]] := axch
```