axiom ac2

Johan G. F. Belinfante
2004 October 28

```
In[1]:= SetDirectory["i:"]; << goedel62.28a; << tools.m;

        :Package Title: goedel62.28a          2004 October 28 at 8:30 p.m.

        It is now:  2004 Oct 28 at 22:51

        Loading Simplification Rules

        TOOLS.M                       Revised 2004 October 28

        weightlimit = 40
```

summary

It is shown that two set forms of the axiom of choice are equivalent. The equivalence amounts to **ac1 <=> ac2**, using the terminology in the following reference (although some liberties have been taken to simplify the statements).

```
In[2]:= "Jean E. Rubin, Set Theory for the Mathematician,
           Holden-Day, San Francisco, 1967. (see pages 78 and 80)";
```

The axiom **ac1** is called **axch** in the **GOEDEL** program, and after the equivalence has been established, the new rewrite rules derived here will convert the statement of axiom **ac2** to **axch** as well, so there is little point in introducing these names formally.

rules for map

```
In[3]:= AssInt[map[x, y], P[cart[V, V]], P[z]]

Out[3]= intersection[map[x, y], P[composite[Id, z]]] == intersection[map[x, y], P[z]]

In[4]:= intersection[map[x_, y_], P[composite[Id, z_]]] := intersection[map[x, y], P[z]]
```

Lemma.

```
In[5]:= intersection[map[intersection[x, y], V], P[cart[y, V]]] // Normality

Out[5]= intersection[map[intersection[x, y], V], P[cart[y, V]]] ==
         map[intersection[x, y], V]

In[6]:= intersection[map[intersection[x_, y_], V], P[cart[y_, V]]] :=
         map[intersection[x, y], V]
```

Lemma.

```
In[7]:= AssInt[map[intersection[x, y], V], P[cart[y, V]], P[z]]

Out[7]= intersection[map[intersection[x, y], V], P[composite[z, id[y]]]] ==
         intersection[map[intersection[x, y], V], P[z]]

In[8]:= intersection[map[intersection[x_, y_], V], P[composite[z_, id[y_]]]] :=
         intersection[map[intersection[x, y], V], P[z]]

In[9]:= member[x, image[IMAGE[FIRST], intersection[FUNS, y]]] // AssertTest

Out[9]= member[x, image[IMAGE[FIRST], intersection[FUNS, y]]] ==
         not[equal[0, intersection[y, map[x, V]]]]

In[10]:= member[x_, image[IMAGE[FIRST], intersection[FUNS, y_]]] :=
          not[equal[0, intersection[y, map[x, V]]]]
```

---

cross-sections

The following temporary terminology is helpful for enunciating the axioms. We shall say that **x** is a **cross-section** of **y** if **x** is a function contained in **y**, with the same domain. The class of all cross-sections of **y** is

```
In[11]:= class[x, and[FUNCTION[x], subclass[x, y], equal[domain[x], domain[y]]]]

Out[11]= X[y]
```

The class **X[x]** of all cross-sections of **x** is sometimes called the cartesian product of the vertical sections of **x**.

---

a special case of interest

Lemma.

```
In[13]:= Assoc[inverse[E], id[complement[singleton[0]]], id[x]]

Out[13]= composite[inverse[E], id[intersection[x, complement[singleton[0]]]]] ==
          composite[inverse[E], id[x]]

In[14]:= composite[inverse[E], id[intersection[x_, complement[singleton[0]]]]] :=
          composite[inverse[E], id[x]]
```

A special case of interest:

```
In[20]:= SubstTest[intersection, map[domain[y], V],
          P[y], y → composite[inverse[E], id[x]]]

Out[20]= intersection[map[intersection[x, complement[singleton[0]]], V],
           P[inverse[E]]] == X[composite[inverse[E], id[x]]]

In[21]:= intersection[map[intersection[x, complement[singleton[0]]], V],
           P[inverse[E]]] := X[composite[inverse[E], id[x]]]
```

---

axiom ac2

The axiom **ac2** says that a cross-section exists for any set **y**. The axiom **axch = ac1** implies that the restriction of **inverse[E]** to any set has a cross-section.

```
In[23]:= image[inverse[IMAGE[id[complement[singleton[0]]]]], image[IMAGE[FIRST],
            intersection[FUNS, P[inverse[E]]]]] // Normality // Reverse

Out[23]= fix[composite[S, IMAGE[FIRST], id[intersection[FUNS, P[inverse[E]]]],
           inverse[IMAGE[id[cart[complement[singleton[0]], V]]]],
           inverse[IMAGE[FIRST]], S, IMAGE[id[complement[singleton[0]]]]]] ==
         image[inverse[IMAGE[id[complement[singleton[0]]]]],
          image[IMAGE[FIRST], intersection[FUNS, P[inverse[E]]]]]

In[24]:= % /. Equal → SetDelayed
```

The axiom **axch = ac1** can be restated as follows:

```
In[25]:= assert[forall[y, not[empty[X[composite[inverse[E], id[y]]]]]]]

Out[25]= axch
```

Axiom **ac2** says:

```
In[26]:= assert[forall[y, not[empty[X[y]]]]]

Out[26]= equal[V, fix[composite[S, id[FUNS], inverse[IMAGE[FIRST]], IMAGE[FIRST]]]]
```

ac2 => ac1

```
In[29]:= SubstTest[implies, and[
           equal[V, fix[composite[S, id[FUNS], inverse[IMAGE[FIRST]], IMAGE[FIRST]]]],
           member[y, V]], not[equal[0, X[y]]], y → composite[inverse[E], id[x]]]

Out[29]= or[not[equal[0, X[composite[inverse[E], id[x]]]]],
           not[equal[V, fix[composite[S, id[FUNS], inverse[IMAGE[FIRST]],
               IMAGE[FIRST]]]], not[member[x, V]]] == True

In[30]:= (% /. x → x_) /. Equal → SetDelayed
```

The variable **x** can now be eliminated.

```
In[31]:= Map[equal[V, #] &, SubstTest[class, x,
            or[member[0, x], not[equal[0, X[composite[inverse[E], id[x]]]]],
             not[equal[V, y]], not[member[x, V]]], y -> fix[
              composite[S, id[FUNS], inverse[IMAGE[FIRST]], IMAGE[FIRST]]]]] // Reverse

Out[31]= or[axch, not[equal[V,
            fix[composite[S, id[FUNS], inverse[IMAGE[FIRST]], IMAGE[FIRST]]]]] == True

In[32]:= % /. Equal → SetDelayed
```

the idea for the proof of the reverse implication

The idea is to show that, for any set **x**, if **y** is a cross-section of **composite[inverse[E], id[P[range[x]]]]**, then **composite[y,VERTSECT[x]]** is a cross-section of **x**.

a general theorem

```
In[33]:= SubstTest[implies, subclass[x, u],
          subclass[composite[x, v], composite[u, v]], {u → inverse[E], v → VERTSECT[y]}]

Out[33]= or[not[subclass[x, inverse[E]]],
           subclass[composite[x, VERTSECT[y]], thinpart[y]]] == True

In[34]:= or[not[subclass[x_, inverse[E]]],
           subclass[composite[x_, VERTSECT[y_]], thinpart[y_]]] := True
```

Corollary.

```
In[35]:= Map[not,
          SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 → subclass[x, inverse[E]],
            p2 -> subclass[composite[x, VERTSECT[y]], thinpart[y]],
            p3 -> subclass[composite[x, VERTSECT[y]], y]}]]

Out[35]= or[not[subclass[x, inverse[E]]],
           subclass[composite[x, VERTSECT[y]], y]] == True

In[36]:= or[not[subclass[x_, inverse[E]]],
           subclass[composite[x_, VERTSECT[y_]], y_]] := True
```

sethood

```
In[37]:= Map[not, SubstTest[and, implies[p1, p2], implies[p3, p4],
           implies[and[p2, p4], p5], not[implies[and[p1, p3], p5]],
           {p1 → member[x, V], p2 → member[thinpart[x], V], p3 → subclass[y, inverse[E]],
            p4 → subclass[composite[y, VERTSECT[x]], thinpart[x]],
            p5 → member[composite[y, VERTSECT[x]], V]}]]

Out[37]= or[member[composite[y, VERTSECT[x]], V],
           not[member[x, V]], not[subclass[y, inverse[E]]]] == True

In[38]:= or[member[composite[y_, VERTSECT[x_]], V],
           not[member[x_, V]], not[subclass[y_, inverse[E]]]] := True

In[39]:= IminComp[id[complement[singleton[0]]], VERTSECT[x], y] // Reverse

Out[39]= image[inverse[VERTSECT[x]], intersection[y, complement[singleton[0]]]] ==
          intersection[domain[x], image[inverse[VERTSECT[x]], y]]

In[40]:= image[inverse[VERTSECT[x_]], intersection[y_, complement[singleton[0]]]] :=
          intersection[domain[x], image[inverse[VERTSECT[x]], y]]
```

conditional rules for composites of functions

Two conditional rules are derived concerning composites of functions.

```
In[41]:= implies[FUNCTION[y], or[FUNCTION[composite[x, y]], not[FUNCTION[x]]]]

Out[41]= True
```

This justifies adding a conditional rewrite rule:

```
In[42]:= or[FUNCTION[composite[x_, y_]], not[FUNCTION[x_]]] := True /; FUNCTION[y]
```

Similar results apply for composites in the reverse order:

```
In[43]:= implies[FUNCTION[x], or[FUNCTION[composite[x, y]], not[FUNCTION[y]]]]

Out[43]= True
```

A second conditional rewrite rule covers this case:

```
In[44]:= or[FUNCTION[composite[x_, y_]], not[FUNCTION[y_]]] := True /; FUNCTION[x]
```

---

hiding the construction

```
In[45]:= implies[
           and[member[x, V], member[y, X[ composite[inverse[E], id[P[range[x]]]]]]],
           member[composite[y, VERTSECT[x]], X[x]]] // NotNotTest

Out[45]= or[and[equal[domain[x], image[inverse[VERTSECT[x]], domain[y]]],
            FUNCTION[composite[y, VERTSECT[x]]], member[composite[y, VERTSECT[x]], V],
            subclass[composite[y, VERTSECT[x]], x]],
           not[equal[domain[y], intersection[complement[singleton[0]], P[range[x]]]]],
           not[FUNCTION[y]], not[member[x, V]], not[member[y, V]],
           not[subclass[y, cart[P[range[x]], V]]], not[subclass[y, inverse[E]]]] == True

In[46]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed

In[47]:= SubstTest[implies, member[z, w], not[equal[0, w]],
          {z -> composite[y, VERTSECT[x]], w -> X[x]}]

Out[47]= or[not[equal[0, X[x]]],
           not[equal[domain[x], image[inverse[VERTSECT[x]], domain[y]]]],
           not[FUNCTION[composite[y, VERTSECT[x]]]],
           not[member[composite[y, VERTSECT[x]], V]],
           not[subclass[composite[y, VERTSECT[x]], x]]] == True

In[48]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed

In[49]:= Map[not,
          SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]], {p1 ->
             and[member[x, V], member[y, X[ composite[inverse[E], id[P[range[x]]]]]]],
            p2 -> member[composite[y, VERTSECT[x]], X[x]], p3 -> not[empty[X[x]]]}]]

Out[49]= or[not[equal[0, X[x]]],
           not[equal[domain[y], intersection[complement[singleton[0]], P[range[x]]]]],
           not[FUNCTION[y]], not[member[x, V]], not[member[y, V]],
           not[subclass[y, cart[P[range[x]], V]]], not[subclass[y, inverse[E]]]] == True

In[50]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

eliminating the variables

The variable **y** can now be eliminated:

```
In[51]:= Map[equal[V, #] &,
           SubstTest[class, y, implies[and[member[x, V], member[y, u]], not[empty[v]]],
            {u -> X[composite[inverse[E], id[P[range[x]]]]], v -> X[x]}]] // Reverse

Out[51]= or[equal[0, X[composite[inverse[E], id[P[range[x]]]]]],
           not[equal[0, X[x]]], not[member[x, V]]] == True

In[52]:= (% /. x → x_) /. Equal → SetDelayed

In[53]:= SubstTest[implies, and[member[x, u], equal[u, v]], member[x, v],
          {u → V, v -> image[inverse[IMAGE[id[complement[singleton[0]]]]],
             image[IMAGE[FIRST], intersection[FUNS, P[inverse[E]]]]]}]

Out[53]= or[not[axch],
           not[equal[0, X[composite[inverse[E], id[x]]]], not[member[x, V]]] == True

In[54]:= (% /. x → x_) /. Equal → SetDelayed
```

The following variant is identical, except for replacing **x** with a power set.

```
In[56]:= SubstTest[implies, and[axch, member[y, V]],
          not[equal[0, X[composite[inverse[E], id[y]]]], y → P[x]]

Out[56]= or[not[axch],
           not[equal[0, X[composite[inverse[E], id[P[x]]]]], not[member[x, V]]] == True

In[57]:= (% /. x → x_) /. Equal → SetDelayed
```

Combining the above results, one obtains the following statement, which hides the construction.

```
In[58]:= Map[not, SubstTest[and, implies[p2, p3], implies[and[p1, p3], p4],
           implies[and[p2, p4], p5], not[implies[and[p1, p2], p5]],
           {p1 → axch, p2 → member[x, V], p3 → member[range[x], V],
            p4 → not[equal[0, X[composite[inverse[E], id[P[range[x]]]]]]],
            p5 → not[equal[0, X[x]]]}]]

Out[58]= or[not[axch], not[equal[0, X[x]]], not[member[x, V]]] == True

In[59]:= (% /. x → x_) /. Equal → SetDelayed
```

Finally, the variable **x** is eliminated:

```
In[60]:= Map[equal[V, #] &, SubstTest[class, x,
           implies[and[p, member[x, V]], not[equal[0, X[x]]]], p → axch]] // Reverse

Out[60]= or[equal[V, fix[composite[S, id[FUNS], inverse[IMAGE[FIRST]], IMAGE[FIRST]]]],
          not[axch]] == True

In[61]:= % /. Equal → SetDelayed
```

The implication now goes both ways:

```
In[62]:= equiv[equal[V,
           fix[composite[S, id[FUNS], inverse[IMAGE[FIRST]], IMAGE[FIRST]]]], axch]

Out[62]= True
```

The following rewrite rule asserts the equivalence **ac1 <=> ac2**.

```
In[63]:= equal[V,
           fix[composite[S, id[FUNS], inverse[IMAGE[FIRST]], IMAGE[FIRST]]]] := axch

In[64]:= fix[composite[S, id[FUNS], inverse[IMAGE[FIRST]], IMAGE[FIRST]]] := V /; axch
```

---

an invariance property

Lemma.

```
In[65]:= image[inverse[IMAGE[id[cart[V, V]]]], fix[
           composite[S, id[FUNS], inverse[IMAGE[FIRST]], IMAGE[FIRST]]]] // Normality

Out[65]= image[inverse[IMAGE[id[cart[V, V]]]],
           fix[composite[S, id[FUNS], inverse[IMAGE[FIRST]], IMAGE[FIRST]]]] ==
         fix[composite[S, id[FUNS], inverse[IMAGE[FIRST]], IMAGE[FIRST]]]

In[66]:= image[inverse[IMAGE[id[cart[V, V]]]],
           fix[composite[S, id[FUNS], inverse[IMAGE[FIRST]], IMAGE[FIRST]]]] :=
         fix[composite[S, id[FUNS], inverse[IMAGE[FIRST]], IMAGE[FIRST]]]
```