SELECT

Johan G. F. Belinfante
2004 November 2

```
In[1]:= SetDirectory["i:"]; << goedel62.31a; << tools.m

        :Package Title: goedel62.31a         2004 October 31 at 11:15 a.m.

        It is now:  2004 Nov 2 at 9:10

        Loading Simplification Rules

        TOOLS.M                       Revised 2004 October 28

        weightlimit = 40
```

summary

The class **SELECT** is defined to be the class of all sets that admit cross-sections. If the axiom of choice holds, then **SELECT** is the universal class. Some properties of **SELECT** are derived that hold whether or not one assumes the axiom of choice to be true. For example, one does not need the axiom of choice to show that every finite set admits a cross-section.

definition

The class **SELECT** is defined by this membership rule:

```
In[2]:= member[x_, SELECT] := and[member[x, V], not[equal[0, X[x]]]]
```

connections with the axiom of choice

Before normalizing **SELECT**, two existing rewrite rules will be rephrased using this new terminology. The first says that the set form **axch** of the axiom of choice is equivalent to the statement that every set has a cross-section:

```
In[3]:= equal[V, SELECT] // AssertTest

Out[3]= equal[SELECT, V] == axch
```

```
In[4]:= equal[SELECT, V] := axch
```

Another statement equivalent to **axch** is that the restriction of the inverse membership relation to any set has a cross-section:

```
In[5]:= subclass[RS[inverse[E]], SELECT] // AssertTest

Out[5]= subclass[RS[inverse[E]], SELECT] == axch

In[6]:= subclass[RS[inverse[E]], SELECT] := axch
```

---

normalization

```
In[7]:= SELECT // Normality // Reverse

Out[7]= fix[composite[S, id[FUNS], inverse[IMAGE[FIRST]], IMAGE[FIRST]]] == SELECT

In[8]:= fix[composite[S, id[FUNS], inverse[IMAGE[FIRST]], IMAGE[FIRST]]] := SELECT
```

---

subclass[X[x], X[y]]

In this section, it is shown that **X[x]** is a subclass of **X[y]** if and only if **composite[Id, x]** is a subclass of **y**, and the domains of **x** and **y** are equal. The implication in one direction is easy:

```
In[9]:= (Map[equal[V, #] &, SubstTest[class, w,
           implies[and[member[w, u], subclass[z, y], equal[domain[z], domain[y]]],
            member[w, v]], {u → X[z], v → X[y]}]] // Reverse) /. z → composite[Id, x]

Out[9]= or[not[subclass[composite[Id, x], y]],
         not[subclass[domain[y], domain[x]]], subclass[X[x], X[y]]] == True

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

The inclusion for the domains can be replaced by equality:

```
In[11]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[and[p2, p3], p4], not[implies[and[p1, p3], p4]],
           {p1 → equal[domain[x], domain[y]], p2 → subclass[domain[y], domain[x]],
            p3 → subclass[composite[Id, x], y], p4 → subclass[X[x], X[y]]}]]

Out[11]= or[not[equal[domain[x], domain[y]]],
           not[subclass[composite[Id, x], y]], subclass[X[x], X[y]]] == True

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

It will be shown that the converse also holds, assuming that **X[x]** is not empty. Part of this is immediate:

```
In[13]:= implies[subclass[X[x], X[y]],
           or[equal[X[x], 0], subclass[composite[Id, x], y]]] // AssertTest

Out[13]= or[equal[0, X[x]], not[subclass[X[x], X[y]]],
           subclass[composite[Id, x], y]] == True

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

Lemma.

```
In[15]:= Map[or[equal[domain[w], domain[y]], #] &, SubstTest[implies,
           and[member[w, u], subclass[u, v]], member[w, v], {u → X[x], v → X[y]}]]

Out[15]= or[equal[domain[w], domain[y]],
           not[equal[domain[w], domain[x]]], not[FUNCTION[w]], not[member[w, V]],
           not[subclass[w, x]], not[subclass[X[x], X[y]]]] == True

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

The equality of **domain[x]** and **domain[y]** follows from this:

```
In[17]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[and[p1, p3], p4], not[implies[and[p1, p2], p4]],
           {p1 → member[w, X[x]], p2 → subclass[X[x], X[y]],
            p3 → equal[domain[w], domain[y]], p4 → equal[domain[x], domain[y]]}]]

Out[17]= or[equal[domain[x], domain[y]],
           not[equal[domain[w], domain[x]]], not[FUNCTION[w]], not[member[w, V]],
           not[subclass[w, x]], not[subclass[X[x], X[y]]]] == True

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

Eliminating the variable **w** yields a simpler result:

```
In[19]:= Map[equal[V, #] &,
           SubstTest[class, w, implies[and[member[w, u], subclass[u, v]],
             equal[domain[x], domain[y]]], {u → X[x], v → X[y]}]] // Reverse

Out[19]= or[equal[0, X[x]],
           equal[domain[x], domain[y]], not[subclass[X[x], X[y]]]] == True

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

This establishes the promised equivalence:

```
In[21]:= equiv[subclass[X[x], X[y]], or[equal[X[x], 0], and[
             subclass[composite[Id, x], y], equal[domain[x], domain[y]]]]] // not // not

Out[21]= True
```

This fact is now made into a new rewrite rule:

```
In[22]:= subclass[X[x_], X[y_]] := or[and[equal[domain[x], domain[y]],
            subclass[composite[Id, x], y]], equal[0, X[x]]]
```

Corollary. If a class admits no cross-section, then neither does any subclass with the same domain.

```
In[23]:= Map[implies[#, and[equal[0, X[x]], equal[0, X[y]]]] &,
          SubstTest[and, subclass[u, v], equal[0, v], {u → X[x], v → X[y]}]]

Out[23]= or[equal[0, X[x]], not[equal[0, X[y]]],
           not[equal[domain[x], domain[y]]], not[subclass[composite[Id, x], y]]] == True

In[24]:= or[equal[0, X[x_]], not[equal[0, X[y_]]], not[equal[domain[x_], domain[y_]]],
           not[subclass[composite[Id, x_], y_]]] := True
```

---

an application

The result of the preceding theorem will be used to show that every set is a subset of one with a cross-section. This is intuitively clear because one can keep enlarging any set, without changing its domain, until it contains a cartesian product with the same domain, and at this point there will exist a constant function which is a cross-section. To carry out this argument one needs the following result about cartesian products:

```
In[25]:= equal[0, X[cart[x, y]]] // AssertTest

Out[25]= equal[0, X[cart[x, y]]] == and[not[equal[0, y]], not[member[x, V]]]

In[26]:= equal[0, X[cart[x_, y_]]] := and[not[equal[0, y]], not[member[x, V]]]
```

This result and the results of the preceding section imply the following lemma.

```
In[27]:= SubstTest[or, equal[0, X[x]],
           not[equal[0, X[y]]], not[equal[domain[x], domain[y]]],
           not[subclass[composite[Id, x], y]], {x → cart[domain[u], range[u]],
            y → union[u, cart[domain[u], range[u]]]}] // MapNotNot

Out[27]= or[not[equal[0, X[union[u, cart[domain[u], range[u]]]]]],
           not[member[domain[u], V]]] == True
```

```
In[28]:= (% /. u → u_) /. Equal → SetDelayed
```

The following lemma makes it easier to eliminate variables.

```
In[29]:= SubstTest[implies, and[subclass[u, v], member[v, w]],
          member[u, image[inverse[S], w]], w → SELECT]

Out[29]= or[equal[0, X[v]], member[u, image[inverse[S], SELECT]],
           not[member[v, V]], not[subclass[u, v]]] == True

In[30]:= (% /. {u → u_, v → v_}) /. Equal → SetDelayed
```

The variables can all be eliminated:

```
In[31]:= Map[class[u, #] &,
          Map[not, SubstTest[and, implies[p1, p2], implies[p3, p4], implies[and[p1, p3],
              p5], implies[and[p2, p3, p5], p6], implies[and[p4, p6], p7], not[
              implies[and[p1, p3], p7]], {p1 → member[u, V], p2 → member[domain[u], V],
              p3 → equal[v, union[u, cart[domain[u], range[u]]]],
              p4 → subclass[u, v], p5 → member[v, V], p6 → member[v, SELECT],
              p7 → member[u, image[inverse[S], SELECT]]}]] /.
           v :> union[u, cart[domain[u], range[u]]]]

Out[31]= image[inverse[S], SELECT] == V

In[32]:= image[inverse[S], SELECT] := V
```

The image under the subset relation **S** is also **V**. This follows from the fact that **0** is a cross-section of itself.

```
In[33]:= equal[image[S, SELECT], V]

Out[33]= True

In[34]:= image[S, SELECT] := V
```

---

two further corollaries

The following corollary follows directly from the main result of the preceding section.

```
In[35]:= ImageComp[inverse[E], inverse[S], SELECT]

Out[35]= U[SELECT] == V

In[36]:= U[SELECT] := V
```

It also follows that **SELECT** is a proper class, which will come as no surprise.

```
In[37]:= (SubstTest[member, U[x], V, x → SELECT] // Reverse) /. Equal → SetDelayed

In[38]:= member[SELECT, x] // AssertTest

Out[38]= member[SELECT, x] == False

In[39]:= member[SELECT, x_] := False
```

---

inverse image rules

Some other inverse image rulesfor **SELECT** are derived in this section.

```
In[40]:= image[inverse[CART], SELECT] // RelnNormality

Out[40]= image[inverse[CART], SELECT] == cart[V, V]

In[41]:= image[inverse[CART], SELECT] := cart[V, V]
```

Corollary: The cartesian product of any two sets admits a cross-section. This is not surprising; any horizontal section of a cartesian product of sets is a cross-section. Horizontal sections are constant functions.

```
In[42]:= Map[subclass[#, SELECT] &, ImageComp[CART, inverse[CART], SELECT]] // Reverse

Out[42]= subclass[range[CART], SELECT] == True

In[43]:= subclass[range[CART], SELECT] := True

In[44]:= image[inverse[FUNPART], SELECT] // Normality

Out[44]= image[inverse[FUNPART], SELECT] == V

In[45]:= image[inverse[FUNPART], SELECT] := V
```

Corollary: Any (small) function admits a cross-section, namely itself.

```
In[46]:= Map[subclass[#, SELECT] &,
           ImageComp[FUNPART, inverse[FUNPART], SELECT]] // Reverse

Out[46]= subclass[FUNS, SELECT] == True

In[47]:= subclass[FUNS, SELECT] := True
```

The class of cross-sections of **x** and the class of cross-sections of **composite[Id, x]** are the same. If one is empty, so is the other.

```
In[48]:= image[inverse[IMAGE[id[cart[V, V]]]], SELECT] // Normality

Out[48]= image[inverse[IMAGE[id[cart[V, V]]]], SELECT] == SELECT

In[49]:= image[inverse[IMAGE[id[cart[V, V]]]], SELECT] := SELECT

In[50]:= image[inverse[SINGLETON], SELECT] // Normality

Out[50]= image[inverse[SINGLETON], SELECT] == V

In[51]:= image[inverse[SINGLETON], SELECT] := V
```

Corollary. Any singleton has a cross-section. Singletons of ordered pairs are functions; the singleton of any other set has **0** as its only cross-section. This corollary will be used later in the proof that every finite set has a cross-section.

```
In[52]:= Map[subclass[#, SELECT] &,
           ImageComp[SINGLETON, inverse[SINGLETON], SELECT]] // Reverse

Out[52]= subclass[range[SINGLETON], SELECT] == True

In[53]:= subclass[range[SINGLETON], SELECT] := True
```

---

invariance under restriction

Lemma. If **x** is a cross-section of **y**, then the restriction of **x** to **z** is a cross-section of the restriction of **y** to **z**.

```
In[54]:= implies[member[x, X[y]],
           member[composite[x, id[z]], X[composite[y, id[z]]]]] // NotNotTest

Out[54]= or[and[equal[intersection[z, domain[x]], intersection[z, domain[y]]],
            FUNCTION[composite[x, id[z]]], member[image[x, z], V],
            member[intersection[z, domain[x]], V], subclass[composite[x, id[z]], y]],
           not[equal[domain[x], domain[y]]], not[FUNCTION[x]],
           not[member[x, V]], not[subclass[x, y]]] == True

In[55]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

The variable **x** can be eliminated:

```
In[56]:= Map[equal[V, #] &,
           SubstTest[class, x, implies[member[x, u], member[composite[x, id[z]], v]],
            {u → X[y], v → X[composite[y, id[z]]]}]] // Reverse

Out[56]= subclass[image[IMAGE[id[cart[z, V]]], X[y]], X[composite[y, id[z]]]] == True
```

```
In[57]:= subclass[image[IMAGE[id[cart[z_, V]]], X[y_]],
           X[composite[y_, id[z_]]]] := True
```

Lemma.

```
In[58]:= equal[intersection[P[cart[z, V]], X[composite[y, id[z]]]],
          X[composite[y, id[z]]]]

Out[58]= True

In[59]:= intersection[P[cart[z_, V]], X[composite[y_, id[z_]]]] :=
          X[composite[y, id[z]]]
```

It follows that if some restriction of a class has no cross-sections, then neither does the class itself.

```
In[60]:= Map[implies[#, equal[0, X[x]]] &,
          SubstTest[and, subclass[u, v], equal[0, v], {u -> X[x],
            v -> image[inverse[IMAGE[id[cart[y, V]]], X[composite[x, id[y]]]]}]]

Out[60]= or[equal[0, X[x]], not[equal[0, X[composite[x, id[y]]]]]] == True

In[61]:= or[equal[0, X[x_]], not[equal[0, X[composite[x_, id[y_]]]]]] := True
```

Eliminating a variable yields a property of **SELECT**.

```
In[62]:= Map[equal[0, composite[Id, complement[#]]] &,
           SubstTest[class, pair[y, w], implies[equal[w, intersection[y, z]],
             or[equal[0, X[y]], not[equal[0, X[w]]]]], z → cart[x, V]]] // Reverse

Out[62]= subclass[image[IMAGE[id[cart[x, V]]], SELECT], SELECT] == True

In[63]:= subclass[image[IMAGE[id[cart[x_, V]]], SELECT], SELECT] := True
```

---

closure under disjoint unions

In this section it is shown that if two sets with disjoint domains each has a cross-section, then the union of those cross-sections is a cross-section of their union. The strategy in this section is to use **funpart** wrappers to avoid some **FUNCTION** literals. The following lemma is needed to do so.

```
In[64]:= SubstTest[implies,
          and[FUNCTION[x], FUNCTION[y], disjoint[domain[x], domain[y]]],
          FUNCTION[union[x, y]], {x → funpart[u], y → funpart[v]}]

Out[64]= or[FUNCTION[union[funpart[u], funpart[v]]],
           not[equal[0, intersection[domain[funpart[u]], domain[funpart[v]]]]]] == True

In[65]:= or[FUNCTION[union[funpart[u_], funpart[v_]]], not[
            equal[0, intersection[domain[funpart[u_]], domain[funpart[v_]]]]]] := True
```

Lemma.

```
In[66]:= implies[and[member[funpart[u], X[x]], member[funpart[v], X[y]],
            equal[0, intersection[domain[x], domain[y]]]],
           member[union[funpart[u], funpart[v]], X[union[x, y]]]] // NotNotTest

Out[66]= or[and[equal[union[domain[x], domain[y]],
             union[domain[funpart[u]], domain[funpart[v]]]],
            FUNCTION[union[funpart[u], funpart[v]]], subclass[funpart[u], union[x, y]],
            subclass[funpart[v], union[x, y]]],
           not[equal[0, intersection[domain[x], domain[y]]]],
           not[equal[domain[x], domain[funpart[u]]]],
           not[equal[domain[y], domain[funpart[v]]]],
           not[member[funpart[u], V]], not[member[funpart[v], V]],
           not[subclass[funpart[u], x]], not[subclass[funpart[v], y]]] == True

In[67]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Eliminating the variables **u** and **v** is expedited by temporarily turning off the flag for equality substitution.

```
In[68]:= equality = False;

In[69]:= Map[equal[0, composite[Id, complement[#]]] &, SubstTest[class, pair[u, v],
            implies[and[member[funpart[u], r], member[funpart[v], s], equal[0, z]],
             member[union[funpart[u], funpart[v]], t]], {r → X[x], s → X[y],
             t → X[union[x, y]], z -> intersection[domain[x], domain[y]]}]] // Reverse

Out[69]= or[not[equal[0, intersection[domain[x], domain[y]]]],
           subclass[image[CUP, cart[X[x], X[y]]], X[union[x, y]]]] == True

In[70]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed

In[71]:= equality = True;
```

Lemma.

```
In[72]:= or[equal[0, X[x]], equal[0, X[y]], not[equal[0, X[union[x, y]]]],
           not[subclass[image[CUP, cart[X[x], X[y]]], X[union[x, y]]]]] // NotNotTest

Out[72]= or[equal[0, X[x]], equal[0, X[y]], not[equal[0, X[union[x, y]]]],
           not[subclass[image[CUP, cart[X[x], X[y]]], X[union[x, y]]]]] == True

In[73]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed

In[74]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[and[p2, p3], p4], not[implies[and[p1, p3], p4]],
           {p1 -> equal[0, intersection[domain[x], domain[y]]],
            p2 -> subclass[image[CUP, cart[X[x], X[y]]], X[union[x, y]]],
            p3 → equal[0, X[union[x, y]]], p4 → or[equal[0, X[x]], equal[0, X[y]]]}]]

Out[74]= or[equal[0, X[x]], equal[0, X[y]],
           not[equal[0, intersection[domain[x], domain[y]]]],
           not[equal[0, X[union[x, y]]]]] == True

In[75]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

---

general case: SELECT is closed under binary unions

By writing the union of any two classes as the union of two classes with disjoint domains, the results of the preceding section can be used to show that if two classes both admit cross-sections, then so does their union. In this more general setting, the union of a cross-section of one class with a cross-section of the other need not be a cross-section of their union. Instead one needs to take the union of a restriction of the cross-section of one with a restriction of the other. Three lemmas are needed for this, two for restricting and one for joining the restrictions.

Lemma 1.

```
In[76]:= SubstTest[implies, and[subclass[composite[Id, u], v],
           equal[domain[u], domain[v]], equal[0, X[v]]], equal[0, X[u]],
          {u → x, v -> union[x, composite[y, id[domain[x]]]]}]

Out[76]= or[equal[0, X[x]],
           not[equal[0, X[union[x, composite[y, id[domain[x]]]]]]]] == True

In[77]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma 2.

```
In[78]:= SubstTest[implies, and[subclass[composite[Id, u], v],
           equal[domain[u], domain[v]], equal[0, X[v]]], equal[0, X[u]],
          {v → intersection[y, complement[cart[domain[x], V]]],
           u -> composite[y, id[complement[domain[x]]]]}]

Out[78]= or[equal[0, X[composite[y, id[complement[domain[x]]]]]],
           not[equal[0, X[intersection[y, complement[cart[domain[x], V]]]]]] == True

In[79]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma 3.

```
In[80]:= SubstTest[implies, and[equal[0, intersection[domain[u], domain[v]]],
           equal[0, X[union[u, v]]]], or[equal[0, X[u]], equal[0, X[v]]],
          {u -> intersection[y, complement[cart[domain[x], V]]],
           v -> union[x, composite[y, id[domain[x]]]]}]

Out[80]= or[equal[0, X[intersection[y, complement[cart[domain[x], V]]]]],
           equal[0, X[union[x, composite[y, id[domain[x]]]]]],
           not[equal[0, X[union[x, y]]]]] == True

In[81]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem: If the classes **x** and **y** both admit cross-sections, then so does their union.

```
In[82]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[and[p1, p2], p4],
           implies[p4, p5], implies[and[p3, p5], p6] , not[implies[and[p1, p2], p6]] ,
           {p1 → not[equal[0, X[x]]], p2 → not[equal[0, X[y]]],
            p3 → not[equal[0, X[union[x, composite[y, id[domain[x]]]]]]],
            p4 → not[equal[0, X[composite[y, id[complement[domain[x]]]]]]],
            p5 → not[equal[0, X[dif[y, composite[y, id[domain[x]]]]]]],
            p6 → not[equal[0, X[union[x, y]]]]}]]

Out[82]= or[equal[0, X[x]], equal[0, X[y]], not[equal[0, X[union[x, y]]]]] == True

In[83]:= or[equal[0, X[x_]], equal[0, X[y_]], not[equal[0, X[union[x_, y_]]]]] := True
```

Corollary. (Eliminating the variables **x** and **y**.)

```
In[84]:= Map[equal[0, composite[Id, complement[#]]] &,
           SubstTest[class, pair[x, y], implies[and[member[x, z], member[y, z]],
             member[union[x, y], z]], z → SELECT]] // Reverse

Out[84]= subclass[image[CUP, cart[SELECT, SELECT]], SELECT] == True

In[85]:= subclass[image[CUP, cart[SELECT, SELECT]], SELECT] := True
```

Using a form of **FINITE** induction, one obtains the following further corollary: every finite set admits a cross-section.

```
In[86]:= SubstTest[implies, and[member[0, z], subclass[range[SINGLETON], z],
           subclass[image[CUP, cart[z, z]], z]], subclass[FINITE, z], z → SELECT]

Out[86]= subclass[FINITE, SELECT] == True

In[87]:= subclass[FINITE, SELECT] := True
```