an equivalent of axch

Johan G. F. Belinfante
2008 October 25

```
In[1]:= SetDirectory["l:"]; << goedel.08oct24b;<< tools.m

        :Package Title: goedel.08oct24b          2008 October 24 at 11:20 p.m.

        It is now:  2008 Oct 25 at 17:19

        Loading Simplification Rules

        TOOLS.M                                    Revised 2008 October 21

        weightlimit = 40
```

reference

The following version of **axch** is available in the **GOEDEL** program.

```
In[2]:= subclass[RS[inverse[E]], SELECT]

Out[2]= axch
```

In this notebook the following seemingly weaker statement is shown to be equivalent to **axch**.

```
In[3]:= subclass[
          image[IMAGE[composite[id[inverse[E]], inverse[FIRST]]], range[POWER]], SELECT];
```

A statement equivalent to this version of **axch** occurs, for example, on page 15 of the following book.

```
In[4]:= "A. G. Kurosh, Lectures on General
          Algebra, English edition, Chelsea Publishing Co., 1965.";
```

a key simplification rule

The following simplification rule is needed immediately.

```
In[5]:= ImageComp[IMAGE[FIRST], inverse[S], intersection[FUNS, P[x]]]

Out[5]= image[inverse[S], image[IMAGE[FIRST], intersection[FUNS, P[x]]]] ==
         image[IMAGE[FIRST], intersection[FUNS, P[x]]]

In[6]:= image[inverse[S], image[IMAGE[FIRST], intersection[FUNS, P[x_]]]] :=
         image[IMAGE[FIRST], intersection[FUNS, P[x]]]
```

With this rewrite rule in place, the **GOEDEL** program can recognize that **axch** is equivalent to the statement that for any collection **x** of sets, there is a function that picks out a member in each nonempty member of **x**.

```
In[7]:= assert[forall[x, exists[t,
           and[FUNCTION[t], subclass[t, inverse[E]], subclass[dif[x, set[0]], domain[t]]]]]]

Out[7]= axch
```

The statement that Kurosh calls the axiom of choice is similar, but only requires the existence of functions that pick members in each nonempty member of any power set. In other words, he replaces **x** by **P[x]**.

```
In[8]:= assert[forall[x, exists[t,
           and[FUNCTION[t], subclass[t, inverse[E]], subclass[dif[P[x], set[0]], domain[t]]]]]]

Out[8]= subclass[image[IMAGE[id[complement[set[0]]], range[POWER]],
          image[IMAGE[FIRST], intersection[FUNS, P[inverse[E]]]]]
```

a technical simplification rule

Lemma. A technical simplification rule.

```
In[9]:= SubstTest[class, x, member[composite[inverse[E], id[x]], t], t -> SELECT] // Reverse

Out[9]= fix[composite[S, IMAGE[FIRST], id[intersection[FUNS, P[inverse[E]]]],
           inverse[IMAGE[id[cart[complement[set[0]], V]]],
           inverse[IMAGE[FIRST]], S, IMAGE[id[complement[set[0]]]]] ==
         image[inverse[IMAGE[composite[id[inverse[E]], inverse[FIRST]]]], SELECT]

In[10]:= % /. Equal→ SetDelayed
```

Theorem.

```
In[11]:= image[inverse[IMAGE[id[complement[set[0]]]]],
           image[IMAGE[FIRST], intersection[FUNS, P[inverse[E]]]]] // Normality

Out[11]= image[inverse[IMAGE[id[complement[set[0]]]]],
           image[IMAGE[FIRST], intersection[FUNS, P[inverse[E]]]] ==
          image[inverse[IMAGE[composite[id[inverse[E]], inverse[FIRST]]]], SELECT]

In[12]:= image[inverse[IMAGE[id[complement[set[0]]]]],
           image[IMAGE[FIRST], intersection[FUNS, P[inverse[E]]]]] :=
          image[inverse[IMAGE[composite[id[inverse[E]], inverse[FIRST]]]], SELECT]
```

equivalence with axch

Lemma. The Kurosh condition implies that the restriction of **inverse[E]** to any power set admits a cross-section. For convenience the power set here is written as **P[U[x]]**.

```
In[13]:= SubstTest[implies, and[member[u, v], subclass[v, w]], member[u, w],
           {u → P[U[x]], v → range[POWER], w → image[inverse[IMAGE[id[complement[set[0]]]]],
              image[IMAGE[FIRST], intersection[FUNS, P[inverse[E]]]]]}] // Reverse

Out[13]= or[not[equal[0, X[composite[inverse[E], id[P[U[x]]]]]]],
           not[member[x, V]], not[subclass[image[
              IMAGE[composite[id[inverse[E]], inverse[FIRST]]], range[POWER]], SELECT]]] == True

In[14]:= (% /. x → x_) /. Equal→ SetDelayed
```

Lemma. Any collection of sets is a subset of a power set: **x ⊂ P[U[x]]**.

```
In[15]:= SubstTest[implies, not[equal[0, X[t]]], not[equal[0, X[composite[t, id[u]]]]],
           {t -> composite[inverse[E], id[P[U[x]]]], u → x}] // Reverse

Out[15]= or[equal[0, X[composite[inverse[E], id[P[U[x]]]]]],
           not[equal[0, X[composite[inverse[E], id[x]]]]] == True

In[16]:= (% /. x → x_) /. Equal→ SetDelayed
```

Theorem.

```
In[17]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
            {p1 → and[member[x, V], subclass[image[
                 IMAGE[composite[id[inverse[E]], inverse[FIRST]]], range[POWER]], SELECT]],
             p2 -> not[equal[0, X[composite[inverse[E], id[P[U[x]]]]]]],
             p3 -> not[equal[0, X[composite[inverse[E], id[x]]]]]}]] // Reverse

Out[17]= or[not[equal[0, X[composite[inverse[E], id[x]]]]],
           not[member[x, V]], not[subclass[image[
              IMAGE[composite[id[inverse[E]], inverse[FIRST]]], range[POWER]], SELECT]]] == True

In[18]:= (% /. x → x_) /. Equal→ SetDelayed
```

Theorem. Eliminating the variable **x** yields the statement: **Kurosh ⇒ axch**.

```
In[19]:= Map[equal[V, #] &, SubstTest[class, x,
           or[member[composite[inverse[E], id[x]], t], not[member[x, V]], not[subclass[u, v]]],
           {t → SELECT, u -> image[IMAGE[composite[id[inverse[E]], inverse[FIRST]]],
              range[POWER]], v -> SELECT}]]

Out[19]= or[axch,
           not[subclass[image[IMAGE[composite[id[inverse[E]], inverse[FIRST]]], range[POWER]],
             SELECT]]] == True

In[20]:= (% /. x → x_) /. Equal→ SetDelayed
```

Corollary. **Kurosh ⇔ axch**.

```
In[21]:= equiv[subclass[image[
             IMAGE[composite[id[inverse[E]], inverse[FIRST]]], range[POWER]], SELECT], axch]

Out[21]= True
```

```
In[22]:= subclass[image[IMAGE[composite[id[inverse[E]], inverse[FIRST]]], range[POWER]],
           SELECT] := axch
```

Corollary. Another variant encountered earlier.

```
In[23]:= SubstTest[subclass, range[POWER], image[inverse[IMAGE[id[complement[set[0]]]]], t],
          t -> image[IMAGE[FIRST], intersection[FUNS, P[inverse[E]]]]]
```

```
Out[23]= subclass[image[IMAGE[id[complement[set[0]]]], range[POWER]],
           image[IMAGE[FIRST], intersection[FUNS, P[inverse[E]]]]] == axch
```

```
In[24]:= subclass[image[IMAGE[id[complement[set[0]]]], range[POWER]],
           image[IMAGE[FIRST], intersection[FUNS, P[inverse[E]]]]] := axch
```