class form of the axiom of choice

Johan G. F. Belinfante
2004 October 26

```
In[1]:= SetDirectory["i:"]; << goedel62.18a; << tools.m;

        :Package Title: goedel62.18a          2004 October 18 at  12:50 noon

        It is now:  2004 Oct 26 at 9:59

        Loading Simplification Rules

        TOOLS.M                      Revised 2004 September 25

        weightlimit = 40
```

summary

This notebook is concerned with proofs of the equivalence of various formulations of a class form of the axiom of choice. As Gödel points out, this is a very strong form of the axiom of choice since it provides for the simultaneous choice of an element from each nonempty set of the universal class **V**. His proof of the consistency of the axiom of choice applies to this strong form of the axiom.

lemma

The following lemma is needed in the next section.

```
In[2]:= equal[x, complement[singleton[y]]] // AssertTest // Reverse

Out[2]= and[equal[V, union[x, singleton[y]]], not[member[y, x]]] ==
         equal[x, complement[singleton[y]]]

In[3]:= and[equal[V, union[x_, singleton[y_]]], not[member[y_, x_]]] :=
         equal[x, complement[singleton[y]]]
```

definition of AxCh

In this section a particular global version of the axiom of choice is studied, which was called **AxCh** in a paper published by the author (see page 323).

```
In[4]:= "Johan G. F. Belinfante, Computer Proofs in Gödel's Class
           Theory with Equational Definitions for Composite and Cross,
           Journal of Automated Reasoning, vol. 22, pp. 311–339 (1999)";
```

Since the **GOEDEL** program lacks an existential quantifier for proper classes, one can only state the global form of the axiom of choice by introducing a Skolem constant **CHOICE** to denote a choice function. If a choice function does exist, it would not be uniquely determined by the demands placed on it by the axiom of choice.

A definition of a global form **AxCh** of the axiom of choice is introduced so that one can reason about this version of the axiom of choice without having to assume that it is true. The definition is wrapped in **class** to prevent it from being automatically expanded.

```
In[5]:= class[w_, AxCh] := class[w, and[FUNCTION[CHOICE], subclass[CHOICE, inverse[E]],
           equal[domain[CHOICE], complement[singleton[0]]]]]
```

The **simplify** flag is turned off to expedite deducing some basic facts.

```
In[6]:= simplify = False;
```

The following rewrite rules are an immediate consequence of the definition.

```
In[7]:= implies[AxCh, FUNCTION[CHOICE]] // AssertTest

Out[7]= or[FUNCTION[CHOICE], not[AxCh]] == True

In[8]:= or[FUNCTION[CHOICE], not[AxCh]] := True

In[9]:= implies[AxCh, subclass[CHOICE, inverse[E]]] // AssertTest

Out[9]= or[not[AxCh], subclass[CHOICE, inverse[E]]] == True

In[10]:= or[not[AxCh], subclass[CHOICE, inverse[E]]] := True

In[11]:= implies[AxCh, equal[domain[CHOICE], complement[singleton[0]]]] // AssertTest

Out[11]= or[equal[complement[singleton[0]], domain[CHOICE]], not[AxCh]] == True
```

```
In[12]:= or[equal[complement[singleton[0]], domain[CHOICE]], not[AxCh]] := True
```

Conversely:

```
In[13]:= implies[and[FUNCTION[CHOICE], subclass[CHOICE, inverse[E]],
            equal[domain[CHOICE], complement[singleton[0]]]], AxCh] // AssertTest

Out[13]= or[AxCh, not[equal[complement[singleton[0]], domain[CHOICE]]],
          not[FUNCTION[CHOICE]], not[subclass[CHOICE, inverse[E]]]] == True

In[14]:= or[AxCh, not[equal[complement[singleton[0]], domain[CHOICE]]],
          not[FUNCTION[CHOICE]], not[subclass[CHOICE, inverse[E]]]] := True
```

The following corollary is needed later.

```
In[15]:= Map[not, SubstTest[and, implies[p1, p2],
          implies[p2, p3], not[implies[p1, p3]], {p1 → AxCh,
            p2 → subclass[CHOICE, inverse[E]], p3 → subclass[inverse[CHOICE], E]}]]

Out[15]= or[not[AxCh], subclass[inverse[CHOICE], E]] == True

In[16]:= or[not[AxCh], subclass[inverse[CHOICE], E_]] := True
```

Rubin's axiom AC1

In this section it is shown that **AxCh** is equivalent to a simplified form of the global axiom called **AC1** in Jean E. Rubin's book (see page 78). In the **GOEDEL** program it is assumed that every element is a set, whereas Rubin allows for the possibility of ur-elements, which are useful for constructing certain models of set theory

```
In[17]:= "Jean E. Rubin, Set Theory for the
            Mathematician, Holden-Day, San Francisco, 1967.";
```

It is convenient to introduce a temporary predicate. We shall say that **f** is a **choice function** on **x** if it satisfies the following condition:

```
In[18]:= choicefunction[f_, x_] :=
          and[FUNCTION[f], subclass[f, inverse[E]], equal[domain[f], x]]
```

Rubin's axiom **AC1** is the statement that there is a choice function on any nonempty class which does not hold the empty set. It is clear that **AC1 => AxCh** because **Axch** amounts to the statement that **CHOICE** is a choice function on **complement[singleton[0]]**.

```
In[19]:= equiv[AxCh, choicefunction[CHOICE, complement[singleton[0]]]] // not // not

Out[19]= True
```

To show the converse statement, that **AC1** follows from **AxCh**, one needs these lemmas:

```
In[20]:= Map[not,
          SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 → AxCh, p2 → FUNCTION[CHOICE], p3 → FUNCTION[composite[CHOICE, id[x]]]}]]

Out[20]= or[FUNCTION[composite[CHOICE, id[x]]], not[AxCh]] == True

In[21]:= or[FUNCTION[composite[CHOICE, id[x_]]], not[AxCh]] := True

In[22]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
           not[implies[p1, p3]], {p1 → AxCh, p2 → subclass[inverse[CHOICE], E],
            p3 → subclass[composite[id[x], inverse[CHOICE]], E]}]]

Out[22]= or[not[AxCh], subclass[composite[id[x], inverse[CHOICE]], E]] == True

In[23]:= or[not[AxCh], subclass[composite[id[x_], inverse[CHOICE]], E]] := True

In[24]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 → AxCh, p2 → equal[domain[CHOICE], complement[singleton[0]]],
            p3 → implies[not[member[0, x]], subclass[x, domain[CHOICE]]]}]]

Out[24]= or[member[0, x], not[AxCh], subclass[x, domain[CHOICE]]] == True

In[25]:= or[member[0, x_], not[AxCh], subclass[x, domain[CHOICE]]] := True
```

The following establishes the implication **AxCh => AC1**.

```
In[26]:= implies[and[AxCh, not[member[0, x]]],
            choicefunction[composite[CHOICE, id[x]], x]] // not // not

Out[26]= True
```

A technical point: At this juncture it is clear that Rubin's assumption that **x** be non-empty is not needed.

CHOICE is a proper class

Since the domain of **CHOICE** is a proper class, it follows that **CHOICE** itself must be a proper class:

```
In[27]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p2, p3], implies[p3, p4], not[implies[p1, p4]],
           {p1 → AxCh, p2 -> equal[domain[CHOICE], complement[singleton[0]]],
            p3 → not[member[domain[CHOICE], V]], p4 → not[member[CHOICE, x]]}]]

Out[27]= or[not[AxCh], not[member[CHOICE, x]]] == True

In[28]:= or[not[AxCh], not[member[CHOICE, x_]]] := True
```

Gödel's Axiom E

Gödel's class form of the axiom of the axiom of choice, which he called **Axiom E**, is stated on page 6 of his monograph.

```
In[29]:= "Kurt Gödel, The Consistency of the Axiom of Choice and of the
            Generalized Continuum Hypothesis with the Axioms of Set Theory,
            Princeton University Press, Princeton, New Jersey, 1940.";
```

A word on notation is in order: Gödel's notation for ordered pairs is the reverse of ours; what we call **pair[x,y]** is written as **<yx>** in his monograph. Also, his use of lowercase letters for sets and uppercase letters for classes is not followed here; we make no notational distinction between sets and classes.   On page 5, he defines a class **x** to be **singlevalued** if

```
In[30]:= assert[forall[u, v, w,
           implies[and[member[pair[u, v], x], member[pair[u, w], x]], equal[v, w]]]]

Out[30]= FUNCTION[composite[Id, x]]
```

Gödel's **Axiom E** says that there exists a class **A** such that

```
In[31]:= and[FUNCTION[composite[Id, A]], assert[forall[x, implies[
             not[empty[x]], exists[y, and[member[y, x], member[pair[x, y], A]]]]]]]

Out[31]= and[equal[V, union[fix[composite[E, A]], singleton[0]]],
          FUNCTION[composite[Id, A]]]
```

apply versus APPLY

In his work on applying the automated reasoing program **Otter** to set theory, Art Quaife used a formulation of the global axiom of choice that differs only slightly from that of Gödel.

```
In[32]:= "Art Quaife, Automated Development of Fundamental Mathematical
            Theories, Kluwer Academic Publishers, 1992 (see page 40).";
```

The main difference is that explicit reference to ordered pairs is replaced by function application, defined as (his notation is **x`y**)

```
In[33]:= apply[x, y]

Out[33]= U[image[x, singleton[y]]]
```

The **GOEDEL** program contains a similar constructor **APPLY** which differs from **apply** by replacing union **U** with intersection **A**.

```
In[34]:= A[image[x, singleton[y]]]

Out[34]= APPLY[x, y]
```

Quaife's formulation of the axiom of choice is that there is a function **f** such that **forall[x, or[equal[0,x], member[apply[f,x], x]]]**. A variable-free form can be derived:

```
In[35]:= and[FUNCTION[f],
           assert[ forall[x, or[equal[0, x], member[apply[ff, x], x]]]]] /. ff -> f

Out[35]= and[FUNCTION[f], subclass[domain[f],
           union[fix[composite[E, f]], image[inverse[f], singleton[0]], singleton[0]]],
          subclass[P[complement[singleton[0]]],
           union[fix[composite[E, f]], singleton[0]]]]
```

The reason for the complexity of this variable-free formulation appears to be that the constructor **apply** fails to distinguish the situation that a function has a value of **0** at a point and the situation that the point is not in the domain of the function:

```
In[36]:= equal[0, apply[funpart[x], y]]

Out[36]= or[equal[0, APPLY[funpart[x], y]], not[member[y, domain[funpart[x]]]]]
```

Because of this, the restriction of the function **f** postulated in this form of the axiom of choice to the class of nonempty sets technically need not be a choice function, although it could of course be extended to a genuine choice function by simply adding **pair[x,0]** for each nonempty set **x** that holds **0** but fails to be in the domain of **f**. The proof that Quaife's version of the axiom of choice is nonetheless equivalent to **AxCh** will not be pursued further here, but in the next section a proof of a simpler theorem of a similar nature is given.

a fixed point theorem

In this section a variant of Quaife's axiom is studied, in which **apply** is replaced with **APPLY**. With this small modification, one fairly easily obtains the following variable-free formulation that more closely resembles Gödel's **Axiom E**:

```
In[37]:= and[FUNCTION[f], assert[forall[x,
             or[equal[0, x], member[APPLY[funpart[z], x], x]]]]] /. funpart[z] → f

Out[37]= and[equal[V, union[fix[composite[E, f]], singleton[0]]], FUNCTION[f]]
```

In this statement **f** is a Skolem constant. Note that this statement also does not require that the empty set be excluded from the domain of **f**. It will now be shown that this form of the axiom of choice is equivalent to **AxCh**. To see that the axiom **AxCh** implies this version of the global axiom of choice, the following lemma is helpful.

```
In[38]:= SubstTest[implies, equal[x, z],
          equal[domain[x], domain[z]], z → intersection[x, inverse[y]]]

Out[38]= or[equal[domain[x], fix[composite[y, x]]],
           not[subclass[x, inverse[y]]]] == True

In[39]:= or[equal[domain[x_], fix[composite[y_, x_]]],
           not[subclass[x_, inverse[y_]]]] := True
```

From the following theorem it follows that **AxCh** => modified **Axiom E**, if one identifies **f** with **CHOICE**.

```
In[40]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p1, p3], implies[p2, p4], implies[and[p3, p4], p5],
           not[implies[p1, p5]], {p1 → AxCh, p2 -> subclass[CHOICE, inverse[E]],
            p3 → equal[domain[CHOICE], complement[singleton[0]]],
            p4 → equal[domain[CHOICE], fix[composite[E, CHOICE]]],
            p5 → equal[fix[composite[E, CHOICE]], complement[singleton[0]]]}]]

Out[40]= or[equal[complement[singleton[0]], fix[composite[E, CHOICE]]], not[AxCh]] ==
          True

In[41]:= or[equal[complement[singleton[0]], fix[composite[E, CHOICE]]], not[AxCh]] :=
          True
```

Conversely, the modified **Axiom E** implies **AxCh** if one defines **CHOICE = composite[f, id[complement[singleton[0]]]**. To establish this, some lemmas are useful.

```
In[42]:= SubstTest[implies, subclass[u, v],
          subclass[composite[w, u], composite[w, v]], {u → id[domain[funpart[x]]],
           v → composite[inverse[funpart[x]], inverse[y]], w → funpart[x]}]

Out[42]= or[not[subclass[domain[funpart[x]], fix[composite[y, funpart[x]]]]],
           subclass[funpart[x], inverse[y]]] == True

In[43]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The converse also holds, so one has this logical equivalence:

```
In[44]:= equiv[subclass[domain[funpart[x]], fix[composite[y, funpart[x]]]],
          subclass[funpart[x], inverse[y]]]

Out[44]= True
```

A rewrite rule corresponding to this fact is introduced:

```
In[45]:= subclass[domain[funpart[x_]], fix[composite[y_, funpart[x_]]]] :=
          subclass[funpart[x], inverse[y]]
```

One can remove the **funpart** wrapper, replacing it with a **FUNCTION** literal:

```
In[46]:= SubstTest[implies,
          and[equal[z, funpart[x]], subclass[domain[z], fix[composite[y, z]]]],
          subclass[z, inverse[y]], z → x]

Out[46]= or[not[FUNCTION[x]], not[subclass[domain[x], fix[composite[y, x]]]],
           subclass[x, inverse[y]]] == True

In[47]:= or[not[FUNCTION[x_]], not[subclass[domain[x_], fix[composite[y_, x_]]]],
           subclass[x_, inverse[y_]]] := True
```

This general theorem can be applied to the particular situation at hand by taking **y** to be **E**.

```
In[48]:= SubstTest[implies,
          and[FUNCTION[x], equal[z, domain[x]], subclass[z, fix[composite[y, x]]]],
          subclass[x, inverse[y]], {y → E, z → complement[singleton[0]]}]

Out[48]= or[not[equal[V, union[fix[composite[E, x]], singleton[0]]]],
           not[equal[complement[singleton[0]], domain[x]]],
           not[FUNCTION[x]], subclass[x, inverse[E]]] == True

In[49]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[50]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
          {u -> complement[singleton[0]], v -> fix[composite[E, x]], w → domain[x]}]

Out[50]= or[equal[V, union[domain[x], singleton[0]]],
           not[equal[V, union[fix[composite[E, x]], singleton[0]]]]] == True

In[51]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[52]:= or[equal[complement[singleton[x]], y], member[x, y],
           not[equal[V, union[y, singleton[x]]]]] // NotNotTest

Out[52]= or[equal[y, complement[singleton[x]]],
           member[x, y], not[equal[V, union[y, singleton[x]]]]] == True

In[53]:= or[equal[y_, complement[singleton[x_]]],
           member[x_, y_], not[equal[V, union[y_, singleton[x_]]]]] := True
```

It now follows that the modification of Quaife's axiom of choice implies **AxCh**, establishing their equivalence.

```
In[54]:= Map[not, SubstTest[and, implies[and[p2, p3], p4], implies[and[p1, p3], p5],
           implies[and[p3, p5], p6], implies[and[p4, p5, p6], p7],
           implies[and[p4, p6, p7], AxCh], not[implies[and[p1, p2, p3], AxCh]],
           {p1 → equal[V, union[fix[composite[E, f]], singleton[0]]], p2 -> FUNCTION[f],
            p3 → equal[CHOICE, composite[f, id[complement[singleton[0]]]]],
            p4 → FUNCTION[CHOICE],
            p5 → equal[V, union[fix[composite[E, CHOICE]], singleton[0]]],
            p6 → equal[complement[singleton[0]], domain[CHOICE]],
            p7 → subclass[CHOICE, inverse[E]]}]]

Out[54]= or[AxCh, not[equal[CHOICE, composite[f, id[complement[singleton[0]]]]]],
           not[equal[V, union[fix[composite[E, f]], singleton[0]]]],
           not[FUNCTION[f]]] == True
```

conditional rules

It would be convenient to be able to simply set **AxCh = True** when one does want to assume that this axiom is true. For this to work one would need to supplement the results obtained by such reasoning with conditional rewrite rules.

```
In[55]:= AxCh = True;
```

In many cases, the needed rules are already in the **GOEDEL** program:

```
In[56]:= FUNCTION[CHOICE]

Out[56]= True

In[57]:= subclass[CHOICE, inverse[E]]

Out[57]= True

In[58]:= domain[CHOICE]

Out[58]= complement[singleton[0]]

In[59]:= FUNCTION[composite[CHOICE, id[x]]]

Out[59]= True

In[60]:= member[CHOICE, x]

Out[60]= False

In[61]:= fix[composite[E, CHOICE]]

Out[61]= complement[singleton[0]]
```

```
In[62]:= or[member[0, x], subclass[x, domain[CHOICE]]]

Out[62]= True
```

The following does require a new conditional rewrite rule.

```
In[63]:= SubstTest[subclass, inverse[u], inverse[v], {u -> CHOICE, v → inverse[E]}]

Out[63]= subclass[inverse[CHOICE], E] == True

In[64]:= subclass[inverse[CHOICE], E] := True /; AxCh
```

No separate rule is needed for the composite with **id[x]**.

```
In[65]:= subclass[composite[id[x], inverse[CHOICE]], E]

Out[65]= True
```