Research Horizons: Computer-Assisted Reasoning

Johan G. F. Belinfante
2004 November 3

```
In[1]:= SetDirectory["i:"]; << goedel63.02a; << tools.m
```

```
:Package Title: goedel63.02a         2004 November 2 at 9:35 a.m.

It is now:  2004 Nov 2 at 21:12

Loading Simplification Rules

TOOLS.M                      Revised 2004 November 2

weightlimit = 40
```

```
In[66]:= Begin["Goedel`Private`"];
```

introduction

True automated reasoning programs, such as McCune's program **Otter**, generally engage in a focussed search for proofs by trying to match patterns in the statement of the theorem with lists of available axioms and previously proven theorems. These programs also make good use of demodulator lists, consisting of equations that can be used to simplify statements, and other means of cutting down on the search space. The **GOEDEL** program is not an automated reasoning program, but was created in 1996 with the sole intension of providing assistance in formulating definitions and theorems in a form suitable for use with an automated reasoning program. The core of the **GOEDEL** program is a set of rewrite rules used by Kurt Gödel to prove a metatheorem concerning the existence of classes. Since these rules generally produce complicated expressions, a large number of rewrite rules were added to obtain more tractable output. As time went on, the program was improved to the point that it can be used to routinely verify hand-produced proofs, or to expose errors made in fallacious proofs. The program can be useful for analyzing machine produced proofs, sometimes resulting in the discovery of improvements that can be made, such as eliminating unnecessary hypotheses. In several cases the program was used to analyze proofs in published papers. The most interesting application is its use to discover new facts, especially new rewrite rules that can be used to simplify the expressions generated by an automated reasoning program. In this talk, after a brief overview, some examples are presented of how the program is used. A survey is given of the axiomatic foundations on which the program is based.

sets versus classes

John von Neumann showed that the complicated type system introduced by Bertrand Russell to avoid his famous paradox can be replaced with a simpler system that has just two types: sets and classes. Later on, this was further simplified by allowing sets to be a special type of class, namely, **x** is a set if **x** is a member of some class. The **GOEDEL** program simplifies this statement, eliminating the quantifier:

```
In[2]:= assert[exists[y, member[x, y]]]

Out[2]= member[x, V]
```

Note: The universal class **V** is the class of all sets. The statement **member[x,V]** just says that **x** is a set.

```
In[3]:= class[x, True]

Out[3]= V
```

The class **V** is a primitive undefined class in Gödel's theory whose meaning is described by the axioms.

---

first order logic

The simplest automated reasoning programs use first order logic, which allows quantifiers on class variables, but not on predicates or constructors. The popular Zermelo-Fraenkel (ZF) axioms for set theory, for instance, requires the use of higher order logic. One of the basic assumptions in the ZF theory is that one can form new sets from old ones by adding the requirement that the members of the new set satisfy some additional conditions, which requires quantifying over statements as well as over sets. Kurt Gödel in 1939 reformulated the von Neumann-Bernays version of set theory so that only a finite number of axioms are required, each of which only involves quantifiers over class variables. Some of the axioms assert that certain specific class constructions are permitted, and others state that certain classes are sets.

class rules

The ability to form classes of sets satisfying any sort of requirement that one wishes to impose is not gone in Gödel's theory, but it has been transferred from the axioms to a meta-theorem, whose proof consists of an algorithm for converting the desired requirement to ones covered by the axioms. The algorithm can be regarded as a parser that analyzes the structure of the sentences in the requirement being added, and converts them into the corresponding class constructions. The idea is to reduce any mathematical statement to an equivalent one in which the sole predicate is **member**. Gödel's constructive proof of this meta-theorem, expressed as a collection of rewrite rules for **class**, forms the basic core of the **GOEDEL** program. Here are two examples of such rules:

```
In[69]:= InfoMatch[class[w_, HoldPattern[exists[y_, p_]]]]
```

```
Out[69]//TableForm=
       class[x_, exists[y_, p_]] := domain[class[pair[x, y], p]]
```

```
In[71]:= InfoMatch[class[w_, HoldPattern[not[p_]]]]
```

```
Out[71]//TableForm=
       class[x_, not[p_]] := intersection[complement[class[x, p]], class[x, True]]
```

The first rule allows one to replace existential quantifiers over sets in statements with **domain** contructors. The algorithm does not permit one to eliminate quantifiers over class variables. In the **GOEDEL** program, the quantifiers **exists** and **forall** are explicitly limited to sets. The sames goes for **class**; one should read **class[x,p]** as the class of all sets **x** for which the statement **p** holds. The second rule allows on to replace negation with **complement**. The intersection with **class[x, True]** is needed in case **x** is not an atomic variable. For example:

```
In[73]:= class[pair[x, y], True]
```

```
Out[73]= cart[V, V]
```

The **GOEDEL** program contains a rewrite rule that converts intersections with cartesian products into composites:

```
In[75]:= intersection[x, cart[V, V]]
```

```
Out[75]= composite[Id, x]
```

A common mistake is to think that **x** and its composite with the identity relation **Id** are always equal. This is only true when **x** is a relation:

```
In[76]:= equal[x, composite[Id, x]]
Out[76]= subclass[x, cart[V, V]]
```

---

classes, constructors and predicates

Consider for example the statement that a class **x** is contained in the power class of the class **y**.

```
In[4]:= subclass[x, P[y]]
Out[4]= subclass[U[x], y]
```

The **GOEDEL** program caused this statement to be rewritten as a new statement, that the sum class of **x** is contained in **y**. In this example, **x** and **y** are variables that stand for any class. Both **U** and **P** are examples of constructors, which form new classes out of old ones. These are not to be confused with functions, because functions are classes, while constructors just provide a notation for a specific method of constructing classes. Thus **P[y]** is just a shorthand for the phrase **power class of y**, and **U[x]** is merely an abbreviation for the phrase **sum class of x**. The predicate **subclass** is used to make a specific statement about a pair of classes. When the variables x and y are replaced with specific classes, further rewrite rules may simplify the results further. For example:

```
In[5]:= subclass[omega, P[omega]]
Out[5]= True
```

Here **omega** is a name for the class of natural numbers, 0, 1, 2, ... In these two cases, there do exist bonafide functions related to these constructors:

```
In[6]:= class[pair[x, y], equal[y, P[x]]]
Out[6]= POWER

In[7]:= class[pair[x, y], equal[y, U[x]]]
Out[7]= BIGCUP
```

```
In[8]:= Map[FUNCTION, {POWER, BIGCUP}]

Out[8]= {True, True}
```

For other constructors, such as **complement**, there is no corresponding function. In this case, the reason is that the complement of a set is not a set.

```
In[9]:= class[pair[x, y], equal[y, complement[x]]]

Out[9]= 0
```

Functions can only be applied to sets in their domain, whereas constructors can be applied to any class. The constructor **APPLY** applies a function to an argument:

```
In[10]:= assert[forall[x, equal[APPLY[BIGCUP, x], U[x]]]]

Out[10]= True
```

---

the meaning of U[x] and P[y]

In the **GOEDEL** program, most classes and class-constructors are defined by means of membership rules. For example, **x** is a member of the power class of **y** if **x** is a set, and **x** is contained in **y**.

```
In[11]:= member[x, P[y]]

Out[11]= and[member[x, V], subclass[x, y]]
```

There is also a membership rule for the sum class **U[x]**, but it involves a quantifier, and has therefore been wrapped inside a rule for **class**. This membership rule says that **x** is a member of **U[z]** if there is a set **y** such that **member[x, y]** and **member[y, z]**.

```
In[12]:= class[x, exists[y, and[member[x, y], member[y, z]]]]

Out[12]= U[z]
```

Quantifiers over sets can always be eliminated, so in principle, one could have defined the sum class constructor by using an unwrapped membership rule, but in the **GOEDEL** program the rewrite rules happen to go in the opposite direction:

```
In[13]:= not[equal[0, intersection[E, cart[singleton[x], y]]]]

Out[13]= member[x, U[y]]
```

Here **E** is another primitive in Gödel's theory, the membership relation. It is related to the predicate **member** as follows:

```
In[14]:= class[pair[x, y], member[x, y]]
Out[14]= E
```

---

image[x,y]

The constructors **U** and **P** are both closely associated with the membership relation **E**. The connection uses the **image** constructor. The **image** under **x** of a class **y** is

```
In[78]:= class[v, exists[u, and[member[pair[u, v], x], member[u, y]]]]
Out[78]= image[x, y]
```

The definition of the sum class is:

```
In[79]:= image[inverse[E], x]
Out[79]= U[x]
```

The definition of power class is:

```
In[56]:= complement[image[E, complement[x]]]
Out[56]= P[x]
```

There is a corresponding function **IMAGE[x]** defined by the following membership rule:

```
In[89]:= member[pair[x, y], IMAGE[z]]
Out[89]= and[equal[y, image[z, x]], member[x, V], member[y, V]]
```

The **IMAGE** constructors can be used to define many functions; for example:

```
In[90]:= IMAGE[inverse[E]]
Out[90]= BIGCUP
```

vertical sections and VERTSECT[x]

Vertical sections are a special type of image. The vertical section of **x** at a set **y** is **image[x, singleton[y]]**. Vertical sections behave better than images in general. Both preserve unions:

```
In[80]:= image[union[x, y], z]
Out[80]= union[image[x, z], image[y, z]]
```

Vertical sections also preserve intersections and relative complements, but images in general do not.

```
In[81]:= image[intersection[x, y], singleton[z]]
Out[81]= intersection[image[x, singleton[z]], image[y, singleton[z]]]

In[83]:= image[intersection[x, complement[y]], singleton[z]]
Out[83]= intersection[complement[image[y, singleton[z]]], image[x, singleton[z]]]
```

The function **VERTSECT[x]** takes each set to the corresponding vertical section of **x**, provided the latter is a set.

```
In[84]:= member[pair[u, v], VERTSECT[x]]
Out[84]= and[equal[v, image[x, singleton[u]]], member[u, V], member[v, V]]
```

The **VERTSECT** constructor can be used to define many important functions. For example:

```
In[86]:= VERTSECT[inverse[S]]
Out[86]= POWER

In[91]:= VERTSECT[inverse[E]]
Out[91]= Id

In[94]:= VERTSECT[Id]
Out[94]= SINGLETON
```

A relation is **thin** if every vertical section is a set. The axiom of replacement is the statement that all functions are thin. The inverse membership relation **inverse[E]** and

the inverse subset relation **inverse[S]** are important examples of thin relations that are not functions.

```
In[85]:= thin[x]
Out[85]= equal[V, domain[VERTSECT[x]]]
```

Several key rewrite rules apply only to thin relations. One of these concerns composites with **inverse[E]**.

```
In[88]:= composite[thinpart[x], inverse[E]]
Out[88]= composite[inverse[E], IMAGE[thinpart[x]]]
```

Here **thinpart[x]** can be viewed as a generic thin relation:

```
In[92]:= thin[thinpart[x]]
Out[92]= True

In[93]:= equal[x, thinpart[x]]
Out[93]= and[equal[V, domain[VERTSECT[x]]], subclass[x, cart[V, V]]]
```

Gödel's axioms

Gödel's axioms for his class theory contain quantifiers over both sets and classes. The statements of his axioms can only be approximated in the **GOEDEL** program because the program only contains quantifiers over sets. Existential quantifiers over classes must be replaced with Skolem functions and constants, and universal quantifiers are replaced with rewrite rules with free variables. In addition, it was found that Kuratowski's construction of the ordered pair is a major source of complexity in the output of Gödel's algorithm, and for this reason the **GOEDEL** program avoids this construction by reducing statements involving ordered pairs to statements involving **equal** instead of reducing all statements to statements involving **member**. The lack of an explicit construction of **pair** also requires modifying some of the axioms in the B group.

---

axiom group A

A1. Meaning of **subclass**.

```
In[15]:= assert[forall[z, implies[member[z, x], member[z, y]]]]
Out[15]= subclass[x, y]
```

A2. Every thing is a class.

```
In[16]:= subclass[x, V]
Out[16]= True
```

A3. Coextension.

```
In[17]:= and[subclass[x, y], subclass[y, x]]
Out[17]= equal[x, y]
```

A4. Existence and sethood of pairsets.

```
In[18]:= member[x, pairset[y, z]]
Out[18]= or[and[equal[x, y], member[x, V]], and[equal[x, z], member[x, V]]]

In[19]:= member[pair[x, y], V]
Out[19]= True
```

Definition of singleton.

```
In[20]:= pairset[x, x]
Out[20]= singleton[x]
```

---

axiom group B

B1. Membership relation.

```
In[21]:= subclass[E, cart[V, V]]
Out[21]= True

In[22]:= member[x, E]
Out[22]= member[first[x], second[x]]
```

B2. intersection.

```
In[23]:= member[x, intersection[y, z]]
Out[23]= and[member[x, y], member[x, z]]
```

B3. complement.

```
In[24]:= member[x, complement[y]]
Out[24]= and[member[x, V], not[member[x, y]]]
```

B4. domain

```
In[25]:= not[equal[0, intersection[y, cart[singleton[x], V]]]]
Out[25]= member[x, domain[y]]
```

B5. (Modified to eliminate specifics of Kuratowski's construction) Ordered pairs, first and second, cartesian products.

```
In[26]:= member[pair[x, y], V]
Out[26]= True
```

There is no membership rule for pairs, but only an equality rule:

```
In[27]:= equal[pair[u, v], pair[x, y]]
Out[27]= and[equal[singleton[u], singleton[x]], equal[singleton[v], singleton[y]]]

In[28]:= member[pair[u, v], cart[x, y]]
Out[28]= and[member[u, x], member[v, y]]

In[29]:= member[x, cart[y, z]]
Out[29]= and[member[first[x], y], member[second[x], z]]

In[30]:= implies[member[x, cart[y, z]], equal[x, pair[first[x], second[x]]]]
Out[30]= True

In[31]:= equal[first[x], V]
Out[31]= not[member[first[x], V]]

In[32]:= member[second[x], V]
Out[32]= member[first[x], V]

In[33]:= equal[second[x], V]
Out[33]= not[member[first[x], V]]
```

Definition of **union**

```
In[34]:= complement[intersection[complement[x], complement[y]]]
Out[34]= union[x, y]
```

Axiom B6 is omitted because it can be proved as a theorem.

Axiom B7. **rotate**

```
In[35]:= subclass[rotate[x], cart[cart[V, V], V]]
Out[35]= True

In[36]:= member[pair[pair[x, y], z], rotate[w]]
Out[36]= and[member[x, V], member[y, V], member[z, V], member[pair[pair[y, z], x], w]]
```

Axiom B8. **flip**

```
In[37]:= subclass[flip[x], cart[cart[V, V], V]]
Out[37]= True

In[38]:= member[pair[pair[x, y], z], flip[w]]
Out[38]= and[member[x, V], member[y, V], member[z, V], member[pair[pair[y, x], z], w]]
```

Comment. The **GOEDEL** program automatically converts **flip** to a composite with the function **SWAP**.

```
In[95]:= flip[x]
Out[95]= composite[x, SWAP]
```

The following definition of **inverse** allows one to omit axiom B6.

```
In[39]:= domain[flip[cart[x, V]]]
Out[39]= inverse[x]
```

Definition of **range**.

```
In[40]:= domain[inverse[x]]
Out[40]= range[x]
```

Definitions of the constructor **image** and the predicate **invariant**.

```
In[41]:= range[intersection[x, cart[y, V]]]
Out[41]= image[x, y]
```

```
In[42]:= invariant[x, y]
Out[42]= subclass[image[x, y], y]
```

---

axiom group C

Definition of **composite**.

```
In[43]:= domain[intersection[rotate[flip[cart[x, V]]], flip[rotate[cart[y, V]]]]]
Out[43]= composite[x, y]
```

Definition of upperbound constructor **UB[x]**.

```
In[44]:= intersection[cart[V, V], complement[composite[complement[x], inverse[E]]]]
Out[44]= UB[x]
```

Definition of subset relation

```
In[45]:= UB[E]
Out[45]= S
```

Definition of identity relation

```
In[46]:= intersection[S, inverse[S]]
Out[46]= Id
```

Definition of vertical section function.

```
In[47]:= intersection[cart[V, V], complement[composite[E, complement[x]]],
          complement[composite[complement[E], x]]]
Out[47]= VERTSECT[x]
```

Definition of successor

```
In[48]:= union[x, singleton[x]]
Out[48]= succ[x]
```

Definition of successor function

```
In[49]:= VERTSECT[union[Id, inverse[E]]]
Out[49]= SUCC
```

Axiom C1 axiom of infinity (needed for arithmetic and counting)

```
In[50]:= member[omega, V]
Out[50]= True

In[51]:= member[0, omega]
Out[51]= True

In[52]:= invariant[SUCC, omega]
Out[52]= True

In[53]:= implies[and[member[0, x], invariant[SUCC, x]], subclass[omega, x]]
Out[53]= True
```

C2: sum class axiom

```
In[55]:= member[U[x], V]
Out[55]= member[x, V]
```

C3: power set axiom

```
In[57]:= member[P[x], V]
Out[57]= member[x, V]
```

The predicate **FUNCTION** could be defined as follows:

```
In[58]:= and[subclass[x, cart[V, V]], subclass[composite[x, inverse[x]], Id]]
Out[58]= FUNCTION[x]
```

C4. axiom of replacement

```
In[59]:= implies[and[FUNCTION[x], member[y, V]], member[image[x, y], V]]
Out[59]= True
```

Comment: The actual definition of the predicate **FUNCTION** in the **GOEDEL** program is equivalent to the above, but a different statement was adopted because it produces cleaner output:

```
In[68]:= InfoMatch[class[w_, HoldPattern[FUNCTION[x_]]]]
Out[68]//TableForm=
      class[w_, FUNCTION[x_]] := Module[{z = Unique[]}, class[w, and[subclass[x, car
```

axiom D axiom of regularity (set form)

```
In[60]:= assert[forall[x, implies[subclass[x, image[E, x]], equal[0, x]]]]

Out[60]= equal[REGULAR, V]
```

The set form implies the class form. A lemma is needed, which is proved here, and made into a new rewrite rule.

```
In[61]:= SubstTest[implies, equal[REGULAR, V],
          implies[subclass[P[y], y], equal[V, y]], y → complement[x]]

Out[61]= or[equal[0, x], not[equal[0, intersection[x, P[complement[x]]]]],
           not[equal[REGULAR, V]]] == True

In[62]:= or[equal[0, x_], not[equal[0, intersection[x_, P[complement[x_]]]]],
           not[equal[REGULAR, V]]] := True
```

This is the class form:

```
In[63]:= implies[equal[V, REGULAR], implies[subclass[x, image[E, x]], equal[0, x]]]

Out[63]= True
```

axiom E axiom of choice (set form)

```
In[64]:= assert[forall[x, implies[not[member[0, x]], exists[y,
             and[FUNCTION[y], subclass[y, inverse[E]], equal[domain[y], x]]]]]]

Out[64]= axch
```

An equivalent of the axiom of choice:

```
In[65]:= assert[forall[x,
           exists[y, and[FUNCTION[y], subclass[y, x], equal[domain[y], domain[x]]]]]]

Out[65]= axch
```

The set form of the axiom of choice does not imply the class form. Gödel proved that the class form of the axiom of choice is consistent with the other axioms by constructing an inner model using a certain class **L** of constructible sets. The definition of **L** resembles that of the class **REGULAR**, but with the power set being replaced with the set of constructible subsets.