H talk at Calculemus 2001.

Johan G. F. Belinfante
2001 June 5

This notebook was used to prepare the slides for the talk at CALCULEMUS''2001, to be presented 2001 June 21. A few items in this notebook were omitted from the slides to keep the talk within the allotted time.

■ Title, abstract, website.

Discovering Theorems using GOEDEL : a case study

Excerpts from Abstract :

The GOEDEL program, a Mathematica implementation of Goedel's algorithm for class formation, is mainly used to formulate definitions needed for Otter proofs in set theory.

The GOEDEL program can also discover new facts. For example, we discovered :

```
image[inverse[TC], P[x]] == P[H[x]]
```

Complete details of all proofs can be found on the author's website.

http : // www.math.gatech.edu / ~belinfan / research / autoreas /

The plan is not to show how the **Otter** proofs go, but rather to show how the **GOEDEL** program is used.

```
<< goedel52.i2; << tests.m
```

```
:Package Title: GOEDEL52.I2      2001 June 5 at 11:45 a.m.

It is now:  2001 Jun 5 at 17:43

Loading Simplification Rules

TESTS.M         Revised 2001 May 19

weightlimit = 30


Context switch to 'Goedel'Private is needed for ReplaceTest

Just ignore the error message about Unterminated use of BeginPackage

Get::bebal : Unterminated uses of BeginPackage or Begin in << tests.m.
```

■ The nine primitives used in Gödel's axioms.

1.The universal class **V**.

```
class[x, True]
```

```
V
```

2. The membership relation **E**.

```
class[pair[x, y], member[x, y]]
```

```
E
```

3. The complement of a class **x**.

```
class[y, not[member[y, x]]]
```

```
complement[x]
```

4. The domain of **x**.

```
class[u, exists[v, member[pair[u, v], x]]]
```

```
domain[x]
```

5. The ternary relation **flip[x]**.

```
equal[class[pair[pair[u, v], w], member[pair[pair[v, u], w], x]], flip[x]]
```

```
True
```

6. The ternary relation **rotate[x]**.

```
class[pair[pair[u, v], w], member[pair[pair[v, w], u], x]]
```

```
rotate[x]
```

7. The unordered pair.

```
equal[class[w, or[equal[w, x], equal[w, y]]], pairset[x, y]] // assert
```

```
True
```

8. The cartesian product of two classes.

```
class[pair[u, v], and[member[u, x], member[v, y]]]
```

```
cart[x, y]
```

9. The intersection of two classes.

```
class[w, and[member[w, x], member[w, y]]]
```

```
intersection[x, y]
```

■ Derived concepts.

Other quantities in Gödel's class theory can be defined in terms of the primitives. For example:

```
class[pair[u, v], member[pair[v, u], x]]
```

```
inverse[x]
```

The quantity **inverse[x]** is defined in terms of the primitives as follows:

```
domain[flip[cart[x, V]]]
```

```
inverse[x]
```

The **range** can now be defined as

```
domain[inverse[x]]
```

```
range[x]
```

The constructor **image** is described by

```
class[v, exists[u, and[member[pair[u, v], x], member[u, y]]]]
```

```
image[x, y]
```

The **image** can be defined in terms of Gödel's primitives and the derived concept **range** as follows:

```
range[intersection[x, cart[y, V]]]
```

```
image[x, y]
```

The conventional description of the composite of **x** and **y** is:

```
class[pair[u, w], exists[v, and[member[pair[v, w], x], member[pair[u, v], y]]]]
```

```
composite[x, y]
```

The definition of **composite** in terms of Gödel's primitives is:

```
domain[intersection[rotate[flip[cart[x, V]]], flip[rotate[cart[y, V]]]]]
```

```
composite[x, y]
```

The subclass relation **S** is:

```
class[pair[x, y], subclass[x, y]]
```

```
S
```

The definition of **S** in terms of the membership relation **E** is:

```
intersection[cart[V, V], complement[composite[complement[E], inverse[E]]]]
```

```
S
```

Gödel provided an algorithm for converting any **class** description into an expression involving his primitives. But in practice, one generally prefers to work with the derived quantities, like **inverse** and **composite**. The **GOEDEL** program goes beyond Gödel's algorithm by adding some rewrite rules to simplify expressions. Another deviation is that the ordered pair is taken as an additional primitive rather than being defined by Kuratowski's construction, or Quaife's modification thereof.

■ Sum class and power class.

The **GOEDEL** program is mainly used to transform class descriptions to expressions involving Gödel's primitives. In the process all quantifiers over sets are eliminated.

Sum class:

```
class[u, exists[v, and[member[v, x], member[u, v]]]]
```

```
U[x]
```

```
image[inverse[E], x]
```

```
U[x]
```

Power class

```
class[w, subclass[w, x]]
```

```
P[x]
```

```
complement[image[E, complement[x]]]
```

```
P[x]
```

A connection between the sum class and power class.

```
U[P[x]]
```

```
x
```

■ The class FULL of all full sets

The **GOEDEL** program can recognize various different descriptions of a given class. For example, the class **FULL** of all full sets can be described in all these ways:

```
class[x, forall[u, v, implies[and[member[u, v], member[v, x]], member[u, x]]]]
```

```
FULL
```

```
class[x, forall[v, implies[member[v, x], subclass[v, x]]]]
```

```
FULL
```

```
class[x, subclass[U[x], x]]
```

```
FULL
```

The equation for **FULL** that was used as a definition in the **Otter** proofs:

```
complement[range[intersection[E, complement[S]]]]
```

```
FULL
```

■ Examples using assert

The **GOEDEL** program will always eliminate quantifiers over sets when one uses **assert**.

```
?? assert
```

```
assert[p] is a statement equivalent to p  obtained by applying Goedel's algorthm
  to class[w,p].  Applying assert repeatedly sometimes simplifies a statement.
```

```
assert[p_] := Module[{w = Unique[]}, equal[V, class[w, p]]]
```

Here are two examples:

```
forall[x, exists[y, and[member[y, z], member[x, y]]]] // assert
```

```
equal[V, U[z]]
```

```
forall[x, exists[y, and[member[y, z], subclass[x, y]]]] // assert
```

```
equal[V, image[inverse[S], z]]
```

Both of these assertions are true about **FULL**.

```
and[equal[V, U[FULL]], equal[V, image[inverse[S], FULL]]]
```

```
True
```

Using **assert** may sometimes reveal equivalent formulations of statements:

```
subclass[image[x, y], complement[z]] // assert
```

```
subclass[cart[y, z], complement[x]]
```

■ lambda

The function corresponding to the sun class constructor is conveniently obtained using **lambda**:

```
lambda[x, U[x]]
```

```
BIGCUP
```

An alternative to **lambda** is to use **class**, but in practice this is not as convenient, especially for complicated functions.

```
class[pair[x, y], equal[y, U[x]]]
```

```
BIGCUP
```

```
lambda[x, P[x]]
```

```
POWER
```

The formula **U[P[x]]=x** implies a formula for the corresponding functions:

```
composite[BIGCUP, POWER]
```

```
Id
```

One can also use **lambda** for binary functions:

```
lambda[pair[x, y], intersection[x, y]]
```

```
CAP
```

The definition of **lambda** uses the function **VERTSECT**. The definition of **VERTSECT** in terms of Gödel's primitives is given in the writeup of this talk, and will not be repeated here.

```
lambda[v, image[x, singleton[v]]]
```

```
VERTSECT[x]
```

The definition of **lambda** is this:

```
?? lambda
```

```
lambda[x,f[x]] is the function which takes x to f[x], and
  lambda[pair[x,y],f[x,y] is the function that takes pair[x,y] to f[x,y], etc.

lambda[x_, e_] :=
 Module[{y = Unique[]}, composite[VERTSECT[class[pair[x, y], member[y, e]]], id[class[x, True]]]]
```

A simple example:

```
VERTSECT[inverse[S]]
```

```
POWER
```

■ IMAGE[x]

The function **IMAGE[x]** is closely related to **VERTSECT[x]**.

```
lambda[y, image[x, y]]
```

```
IMAGE[x]
```

A simple example:

```
IMAGE[inverse[E]]
```

```
BIGCUP
```

Each of these functions could be defined in terms of the other, but in practice **VERTSECT** is simpler than **IMAGE**.

```
composite[IMAGE[x], SINGLETON]
```

```
VERTSECT[x]
```

```
VERTSECT[composite[x, inverse[E]]]
```

```
IMAGE[x]
```

Their domains are related as follows:

```
domain[IMAGE[x]]
```

```
P[domain[VERTSECT[x]]]
```

An important special case:

```
lambda[y, intersection[x, y]]
```

```
IMAGE[id[x]]
```

This unary function is not to be confused with the binary function **CAP**. When **a** is a set, the binary function **CAP** is related to the unary function **IMAGE[id[a]]** as follows:

```
composite[CAP, RIGHT[a]]
```

```
IMAGE[id[a]]
```

For any set **a** the function **RIGHT[a]** is given by

```
lambda[x, pair[x, a]]
```

```
RIGHT[a]
```

■ Hereditary core

The hereditary core **H[x]** of a class **x** is the largest full subclass of **x**. It is defined as the union of all full subsets of **x**.

```
class[u, exists[v, and[member[u, v], full[v], subclass[v, x]]]]
```

```
H[x]
```

The definition used in our **Otter** work is equivalent:

```
U[intersection[FULL, P[x]]]
```

```
H[x]
```

The corresponding function is called **HC**.

```
lambda[x, H[x]]
```

```
HC
```

The function **HC** is the composite of functions discussed earlier:

```
composite[BIGCUP, IMAGE[id[FULL]], POWER]
```

```
HC
```

When **x** is a proper class, one can still obtain **H[x]** from the function **HC** indirectly:

```
U[image[HC, P[x]]]
```

```
H[x]
```

The transitive closure **tc[x]** of a class **x** is the smallest full class that contains **x**. When **a** is a set, it is given as the intersection of all full sets that contain **a**.

```
A[intersection[FULL, image[S, singleton[a]]]]
```

```
tc[a]
```

The corresponding function is

```
lambda[x, tc[x]]
```

```
TC
```

When **x** is a proper class, the transitive closure **tc[x]** is the union of the class of transitive closures of all subsets of **x**.

```
U[image[TC, P[x]]]
```

```
tc[x]
```

■ Discovering theorems with VSNormality, and other tests.

Using the **GOEDEL** program, many facts about **HC** and **TC** were discovered, for example:

```
composite[inverse[HC], S]
```

```
composite[S, TC]
```

The details of the discovery of the following identity using the **GOEDEL** program can be found in the writeup of this talk:

```
image[inverse[TC], P[x]]
```

```
P[H[x]]
```

Once this fact was discovered, it was not difficult to get an **Otter** proof of it. The basic technique used is **VSNormality**, one of several discovery tools.

```
? VSNormality
```

```
Goedel‘Private‘VSNormality
```

```
VSNormality[x_] := Module[{u = Unique[], v = Unique[]}, member[u, V] = True;
  member[v, V] = True; composite[Id, x] == class[pair[u, v], member[v, image[x, singleton[u]]]]]
```

To use this test, we just apply it to any relation of interest. (When the **GOEDEL** program fails to discover anything, it returns **True**.) Rather than rehashing the discoveries that are mentioned in the writeup, we offer several other simple examples of mildly interesting facts discovered in this fashion:

```
composite[BIGCUP, TC, SINGLETON] // VSNormality
```

```
composite[BIGCUP, TC, SINGLETON] == TC
```

```
composite[complement[E], TC] // VSNormality
```

```
composite[complement[E], TC] == composite[inverse[IMAGE[inverse[TC]]], complement[E]]
```

```
composite[complement[HC], HC] // VSNormality
```

```
composite[complement[HC], HC] == composite[Id, complement[HC]]
```