TC talk at IJCAR 2001.

Johan G. F. Belinfante
2001 June 3

This notebook was used to prepare the slides for the talk at IJCAR'2001, to be presented 2001 June 20. A few items in this notebook were omitted from the slides to keep the talk within the allotted time.

■ Title, abstract, website.

Computer Proofs about transitive closure

Excerpts from Abstract:

We focus on a computer proof that uses a recursive definition for tc[x].

The NBG axioms for class theory are used, as modified by Quaife.

We proved using Otter: every set is a member of a full set: U[FULL] == V

Complete details of the proof 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 was used to cook up the definition that one needs to get started.

```
<< goedel52.h26; << tests.m
```

```
:Package Title: GOEDEL52.H26      2001 June 1 at 6:15 p.m.

It is now:  2001 Jun 3 at 11:5

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.
```

■ Basic warmup exercises: Definitions of domain, range, fix, id and image.

The universal class **V** is the class of all sets.

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

```
V
```

Definitions of **domain**, **range** and **image**.

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

```
domain[x]
```

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

```
range[x]
```

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

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

The class of fixed points of **x**.

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

```
fix[x]
```

The identity relation restricted to a class **x**.

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

```
id[x]
```

Global identity:

```
id[V]
```

```
Id
```

The constructors **fix** and **id** are related as follows:

```
subclass[id[x], y]
```

```
subclass[x, fix[y]]
```

Restriction of a relation:

```
intersection[x, cart[y, z]]
```

```
composite[id[z], x, id[y]]
```

The image is the range of a restriction:

```
range[composite[id[z], x, id[y]]]
```

```
intersection[z, image[x, y]]
```

■ Definitions of sum class, power class.

Sum class:

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

```
U[x]
```

Power class

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

```
P[x]
```

A connection between the sum class and power class.

```
subclass[x, P[y]]
```

```
subclass[U[x], y]
```

■ Definition of full and the class FULL of all full sets

Definition of the predicate **full**.

```
full[x]
```

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

The class of all full sets.

```
class[x, full[x]]
```

```
FULL
```

Why full classes are also called transitive classes:

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

```
FULL
```

Another characterization:

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

```
FULL
```

The class **OMEGA** of ordinal numbers: (one can omit **REGULAR** if the axiom of regularity holds)

```
intersection[REGULAR, FULL, P[FULL]]
```

```
OMEGA
```

One of the theorems that was proved using recursion: ***every set is a member of a full set***.
The **GOEDEL** program now knows this theorem, so it simplifies this statement to **True**.

```
U[FULL] == V
```

```
True
```

Here we write it out using quantifiers, temporarily replacing **FULL** by **z** here to avoid just getting **True** again.

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

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

■ The membership relation and the subclass relation

The membership relation **E** is one of Gödel's primitives.

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

```
E
```

The subclass relation is:

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

```
S
```

Some connections with concepts mentioned earlier.

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

```
U[x]
```

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

```
complement[P[complement[x]]]
```

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

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

```
FULL
```

■ The definition of subvar and its relation to GREATEST.

The cross product of two relations.

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

```
cross[x, y]
```

```
image[cross[x, y], z]
```

```
composite[y, z, inverse[x]]
```

The class of all subvariant sets for a relation **x** is called **subvar[x]**.

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

```
subvar[x]
```

An example discussed at FTP'2000: (Here **FINITE** is the class of all finite sets.)

```
complement[U[subvar[intersection[S, complement[Id]]]]]
```

```
FINITE
```

The union of all subvariant sets is invariant:

```
image[x, U[subvar[x]]] == U[subvar[x]]
```

```
True
```

The upper bound relation **UB[x]** associated with a relation **x**.

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

```
UB[x]
```

A simple example: (This is just a nicer way to write the definition of **S** in terms of **E**.)

```
UB[E]
```

```
S
```

In our **Otter** work, the subclass relation **S** is defined in terms of the membership relation.

```
composite[Id, complement[composite[complement[E], inverse[E]]]]
```

```
S
```

```
composite[Id, complement[composite[complement[x], inverse[E]]]]
```

```
UB[x]
```

The greatest relation **GREATEST[x]** associated with any given relation **x**.

```
intersection[UB[x], inverse[E]]
```

```
GREATEST[x]
```

Connection between **subvar** and **GREATEST**.

```
complement[domain[GREATEST[complement[x]]]]
```

```
subvar[x]
```

■ Building in initial conditions: the basic idea.

One of the nice things about subvariance is that one can build in initial conditions.

```
subclass[x, image[union[id[y], z], x]] // assert
```

```
subclass[x, union[y, image[z, x]]]
```

```
class[x, subclass[x, union[y, image[z, x]]]]
```

```
subvar[union[z, id[y]]]
```

■ Recursive formula for tc[x]

Naively, one wishes to define **tc[x]** as the union of **x**, **U[x]**, **U[U[x]]**, etc., which implies that **tc[x]** is a solution of a recursion relation,

```
tc[x] == union[x, U[tc[x]]]
```

```
True
```

Unforunately, this does not uniquely define **tc[x]** because there are infinitely many such solutions. For example, when **x = 0**, any limit ordinal satisfies this condition, whereas we only want the least solution, **tc[0]=0**.

■ Using omega to build the minimal solution.

To build a minimal solution, we look for relations whose vertical sections are partial solutions of the recursion relation. By replacing **equal** by **subclass** in the recursion relation, one can build in the initial conditions. We define a partial solution **p** to be a relation that belongs to the following **subvar**:

```
member[p, subvar[union[cross[E, inverse[E]], id[cart[V, x]]]]] // assert
```

```
and[member[p, V], subclass[p, union[cart[V, x], composite[inverse[E], p, inverse[E]]]]]
```

Let **n** be some ordinal number, and examine the vertical section at **n**:

```
member[n, V] := True
```

For simplicity, we replace **subclass** by <, and take advantage of the monotonicity of **image**:

```
Map[image[#, singleton[n]] &,
    p < union[cart[V, x], composite[inverse[E], p, inverse[E]]]]
```

```
image[p, singleton[n]] < union[x, U[image[p, n]]]
```

This becomes clearer when one considers the first few cases. Replace **n** by **0**, **succ[0]**, etc. The first case is:

```
(image[p, singleton[n]] < union[x, U[image[p, n]]]) /. n -> 0
```

```
image[p, singleton[0]] < x
```

The next case is this:

```
(image[p, singleton[n]] < union[x, U[image[p, n]]]) /. n -> succ[0]
```

```
image[p, singleton[singleton[0]]] < union[x, U[image[p, singleton[0]]]]
```

Therefore, substituting (by hand) the preceding inequality, we find:

```
image[p, singleton[succ[0]]] < union[x, U[x]]
```

```
image[p, singleton[singleton[0]]] < union[x, U[x]]
```

One can now continue:

```
(image[p, singleton[n]] < union[x, U[image[p, n]]]) /. n -> succ[succ[0]]
```

```
image[p, singleton[union[singleton[0], singleton[singleton[0]]]]] <
 union[x, U[image[p, singleton[0]]], U[image[p, singleton[singleton[0]]]]]
```

Hence, substituting again (by hand), we find:

```
image[p, singleton[succ[succ[0]]]] < union[x, U[x], U[union[x, U[x]]]]
```

```
image[p, singleton[union[singleton[0], singleton[singleton[0]]]]] <
 union[x, U[x], U[U[x]]]
```

In view of this one naturally comes up with the following definition for the transitive closure:

```
tc[x] == image[U[subvar[union[cross[E, inverse[E]], id[cart[V, x]]]], omega];
```

Starting from this definition, we used **Otter** to prove that **tc[x]** is the least full class that contains **x**, and that **tc[x]** is a set when **x** is a set. The proofs of these theorems make use of the following corollary of transfinite induction, which was proved in the author's JAR paper on ordinal numbers.

```
implies[subclass[P[x], x], subclass[OMEGA, x]];
```

The proof of the particular lemma where this fact is used is given in the appendix to the writeup of this talk.
For complete details of all the proofs, see the website.

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

■ Appendix: how one could discover the key subvar formula.

Deriving the **subvar** formula from the partial recursion condition requires an extra rule, without which the **GOEDEL** program yields a mess:

```
class[p, and[subclass[p, cart[V, V]],
  forall[n, subclass[image[p, singleton[n]], union[x, U[image[p, n]]]]]]]

intersection[complement[
  fix[composite[inverse[IMAGE[id[cart[V, complement[x]]]]], inverse[IMAGE[SWAP]],
    E, UB[complement[cross[inverse[E], E]]], IMAGE[SWAP]]]], P[cart[V, V]]]
```

The needed formula is obtained by using **Normality** (and some other tricks):

```
subvar[union[composite[SWAP, x, SWAP], id[cart[V, complement[y]]]]] // Normality //
  InvertFix // Reverse

intersection[
  complement[fix[composite[inverse[IMAGE[id[cart[V, y]]]], inverse[IMAGE[SWAP]],
     E, UB[complement[x]], IMAGE[SWAP]]]], P[cart[V, V]]] ==
 subvar[union[composite[SWAP, x, SWAP], id[cart[V, complement[y]]]]]
```

We add this new rule:

```
intersection[complement[fix[composite[inverse[IMAGE[id[cart[V, y_]]]],
     inverse[IMAGE[SWAP]], E, UB[complement[x_]], IMAGE[SWAP]]]], P[cart[V, V]]] :=
 subvar[union[composite[SWAP, x, SWAP], id[cart[V, complement[y]]]]]
```

Now one can derive the **subvar** formula from the partial recursion condition:

```
class[p, and[subclass[p, cart[V, V]],
  forall[n, subclass[image[p, singleton[n]], union[x, U[image[p, n]]]]]]]

subvar[union[cross[E, inverse[E]], id[cart[V, x]]]]
```