

Computer Proofs about Transitive Closure

Johan Gijsbertus Frederik Belinfante

Georgia Institute of Technology, Atlanta, GA 30332-0160 (U.S.A.)
belinfan@math.gatech.edu

Abstract. As a part of ongoing research on automated reasoning in set theory, we focus here on an example of a computer proof that involves a recursive definition. The transitive closure theorems imply that every set is a member of a full set, which can be shown to be equivalent to the equation $U(\text{FULL}) = V$ in the framework of NBG class theory. We rely on McCune's first-order automated reasoning program `Otter` to obtain the formal proof of this theorem, and use a MathematicaTM implementation of Gödel's algorithm for class formation to help discover and formulate lemmas that are needed.

1 Introduction.

Encouraging progress has been made recently (Belinfante, 1999a, 1999b and 2000b) using McCune's (1994) first order logic program `Otter` for automated reasoning in the von Neumann-Bernays theory of sets and classes, building on prior work by Robert Boyer et al. (1986) and Art Quaipe (1992a, 1992b). While various forms of ordinary and transfinite induction were used in some computer proofs in our previous work, until now none of these used recursive definitions. Proofs using recursion produced by hand often employ metatheoretic arguments that are difficult to handle using first order resolution style theorem provers. The transitive closure theorems imply that every set is a member of a full set, which can be concisely reformulated as the equation $U(\text{FULL}) = V$ in NBG class theory. `Otter`'s proof depends on a lemma in which recursion is used to construct the transitive closure of any class; this lemma employs transfinite induction, but no metatheory.

The key to bringing set theory within the realm of first order logic is Gödel's (1940) finite axiomatization for class theory, which he achieved at the cost of abandoning the usual class formation constructor $\{x \mid p(x)\}$ in favor of building classes in terms of the universal class V , the membership relation E , and seven other basic class constructors: `complement`, `domain`, `flip`, `rotate`, `pairset`, `cart`, `intersection`.

To help prepare input files for proofs in set theory using McCune's automated reasoning program `Otter`, one must convert the customary definitions of classes into expressions built up from the primitive constructors, which we do by using the `GOEDEL` program, an implementation (Belinfante, 1996 and 2000a) in MathematicaTM of an algorithm which Kurt Gödel presented in his proof of his fundamental Class Existence metatheorem schema. Because many rewrite rules were added to produce compact definitions, Gödel's proof of termination no longer applies, but the `GOEDEL` program nonetheless appears to be practical as an ad hoc tool for formulating definitions and simplifying statements of theorems. Although it contains no mechanism for carrying out deductions, the `GOEDEL` program sometimes manages to prove statements by simplifying them to `True`.

Some theorems about transitive closure are listed in appendix A. Further details, including related theorems, proof summaries and other useful information about each of the **Otter** proofs of the theorems about transitive closure (and several thousand other theorems in set theory) are posted on the author's website:

<http://www.math.gatech.edu/~belinfan/research/>.

A recent version of the **GOEDEL** program is also available there.

Generally, the formal proofs obtained with **Otter** tend to lag behind what has been discovered using the **GOEDEL** program, but the **GOEDEL** program is no substitute for **Otter** because it does not produce readable proofs; using Mathematica's built-in **Trace** procedure generally yields a voluminous and nearly unintelligible accounting of what took place. The notations used in **Otter** and in the **GOEDEL** program are similar, but not identical; we use both notations here to help clarify which results were discovered using the **GOEDEL** program, and which have been proved using **Otter**. For brevity, in this talk, an equation $a = b$ is often simply written when Mathematica input **a** produces output **b**, or viceversa, for some version of the **GOEDEL** program. (Facts discovered with one version are often added as rewrite rules in later ones.)

2 Eliminating Skolem functions

Gödel's algorithm is presented in the **GOEDEL** program as a series of definitions for a Mathematica function `class[x,p]`. The first argument **x**, which is assumed to be a set, must be either an atomic symbol, or an expression of the form `pair[u,v]` where **u** and **v** in turn are either atomic symbols or pairs, and so on. The second argument **p** is some statement which can involve the variables that appear in the expression **x**, as well as other variables that may represent arbitrary classes (not just sets). The statement may contain quantifiers, but all quantified variables must be sets. The quantifiers `forall` and `exists` used in the **GOEDEL** program are explicitly restricted to set variables.

The rewrite rules in the **GOEDEL** program can be used to simplify statements as well as descriptions of classes, and in particular, to eliminate quantifiers. Given any statement **p**, one can form the class `class[w,p]` where **w** is any variable that does not occur in the statement **p**. This class is the universal class **V** if **p** is true, and is the empty class when **p** is false. The Mathematica definition

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

thus produces a new statement equivalent to the original one. The occurrence of `class` here causes Gödel's algorithm to be invoked, the meaning of the statement **p** to be interpreted, and the rewrite rules in the **GOEDEL** program to be applied. The transformed statement need not be simpler than the statement one started with, but often it is. To improve readability of the output, further rewrite rules may convert the equations obtained with `assert` back to simpler nonequational statements.

Alfred Tarski and Steven Givant (1987) showed how set theory can be formulated as an equational theory without set variables. It has recently been proposed by Omodeo and Formisano (1998) that this equational formalism be used for automated proofs in set theory. The `assert` mechanism in the **GOEDEL** program achieves much

the same objective. Any statement is converted by `assert` into an equation of the form `equal[V, x]`.

Eliminating quantifiers over set variables helps minimize the number of new Skolem functions that are introduced in converting statements to clause form. To do this, an adequate supply of standard functions and constructors must be available. For example, by introducing the function `SUCC`, Quaife formulated the condition that the set `omega` of natural numbers is closed under the successor operation without using variables as `subclass(image(SUCC, omega), omega)`. This is one of the techniques he exploited to reduce the plethora of Skolem functions that had plagued earlier work.

3 VERTSECT and thin relations

Replacing the function symbols of first order logic by bonafide set-theoretic functions not only helps to eliminate Skolem functions, but also greatly improves the readability of the statements of many theorems. A standard way to obtain definitions for many functions is in terms of a basic constructor `VERTSECT`, enabling one to define functions by specifying the result obtained when they are applied to an input. The basic idea is not limited to functions; any relation can be specified by giving a formula for its vertical sections. The vertical sections of a relation `z` are the family of classes

$$\text{class}[y, \text{member}[\text{pair}[x, y], z]] = \text{image}[z, \text{singleton}[x]].$$

One way to prove that one relation is contained in another is to show that every vertical section of the one is a subclass of the corresponding vertical section of the other. A simple example is the proof of Theorem `TC-CO-BC`, which states that the functions `TC` and `BIGCUP` commute. The function `TC` maps any set to its transitive closure; the function `BIGCUP` maps any set `x` to its sum class `U(x)`. `Otter` found a 3 step proof of Theorem `TC-CO-BC` by using a Skolemized version of the following corollary of Quaife's Theorem `C05`

```
(all x y z ((all w subclass(image(x, image(y, singleton(w))),
                             image(z, singleton(w))))
           -> subclass(composite(x, y), z))).
```

This same corollary was used by `Otter` in the proof of Theorem `TC-CLOS`, a result which is stated without proof on page 212 of Rubin's book (1967). Theorem `TC-CLOS` asserts that `composite(inverse(E), TC)` is the smallest transitive relation which contains `inverse(E)`. One reason that vertical section arguments are more efficient than working with individual points is that only one Skolem function is needed rather than four, and one avoids having to deal with membership statements and ordered pairs.

Once one comes to realize the usefulness of working with vertical sections instead of points, it becomes natural to introduce the function which assigns these vertical sections:

```
VERTSECT[z] = class[pair[x, y], equal[y, image[z, singleton[x]]]]
```

Gödel's algorithm converts this formula to the expression

```
VERTSECT[z] = composite[Id,intersection[
    complement[composite[E,complement[z]]],
    complement[composite[complement[E],z]]]].
```

For many relations z the vertical sections need not be sets. The domain of the function $\text{VERTSECT}[z]$ in general is the class of all sets x for which $\text{image}[z, \text{singleton}[x]]$ is also a set. We call a relation *thin* when all its vertical sections are sets. The axiom of replacement implies that functions are thin, and the sum class and power class axioms imply that the relations $\text{inverse}[E]$ and $\text{inverse}[S]$ are thin, where E and S are the membership and the subset relations, respectively.

One can use VERTSECT to obtain a formula (Belinfante, 2000a) for any function from a formula for its application $A[\text{image}[f, \text{singleton}[x]]]$. This is done neatly in the GOEDEL program by introducing the Mathematica definition

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

Two simple examples of function definitions obtained using lambda are the function SINGLETON which takes any set to its singleton,

$$\text{lambda}[x, \text{singleton}[x]] = \text{VERTSECT}[\text{Id}] = \text{SINGLETON},$$

and the function POWER which takes any set to its power set $P[x]$,

$$\text{lambda}[x, P[x]] = \text{VERTSECT}[\text{inverse}[S]] = \text{POWER}.$$

In addition to VERTSECT , it is also convenient to introduce the related constructor IMAGE , defined by

```
IMAGE[x] = lambda[u, image[x,u]]
          = VERTSECT[composite[x, inverse[E]]].
```

One of the first applications found for IMAGE was the definition

$$\text{lambda}[x, U[x]] = \text{IMAGE}[\text{inverse}[E]] = \text{BIGCUP}$$

of the function BIGCUP . The constructor IMAGE is not a functor in the category theory sense. The function $\text{IMAGE}(x)$ does not in general preserve composites, but according to Theorem IMG-CO2 , this does hold when the right hand factor is thin. While IMAGE preserves the global identity function according to Theorem IMG-ID , in general $\text{IMAGE}(\text{id}(x))$ is not an identity function, but it is nonetheless a useful function. From the GOEDEL program we learn:

$$\text{lambda}[w, \text{intersection}[x, w]] = \text{IMAGE}[\text{id}[x]].$$

A large group of theorems about IMAGE were in fact proved using Otter quite some time before those about VERTSECT because they are needed to construct certain standard functions (Belinfante, 2000a). For our present purpose, the most important fact is Theorem IMG-VS-8

```
equal(D(IMAGE(x)), P(D(VERTSECT(x))))). %*IMG-VS-8
```

which states that the domain of $\text{IMAGE}(x)$ is the power class of the domain of $\text{VERTSECT}(x)$.

4 Proofs of the Transitive Closure Theorems

Three groups about transitive closure were proved using `Otter`. All three TC groups have been placed immediately following the groups of theorems (Belinfante, 2000b) about `subvar`. The first group, which contains all the hard work, consists of some technical lemmas needed to apply the transfinite induction theorem. The definition of transitive closure is given in the second group, and the transitive closure theorems are all in that group. The third group is concerned with properties of the function `TC` which assigns to each set `x` its transitive closure `tc(x)`.

We comment briefly about each of the technical lemmas. The easy Lemma `TC-LEM1A` which follows readily from Quaipe's Theorem `C05` just helps simplify the proof of Lemma `TC-LEM1B`. The 10 step proof of Lemma `TC-LEM1B` found by `Otter` in 20 seconds uses Lemma `VS-12` about `VERTSECT`, the two facts `TH-E-IN` and `TH-IM` about thin relations, Theorems `IMG-DO` and `IMG-VS-8` about `IMAGE`, as well as Theorem `ADJ-IM1` about the function `ADJOIN(x)`. If `x` is a set, the function `ADJOIN(x)` maps any set `y` to the set `union(x,y)`; when `x` is a proper class, `ADJOIN(x)` is the empty set. Although technically transfinite methods are not required to prove Lemma `TC-LEM1C`, the short 4 step proof found by `Otter` used the following useful Corollary (Belinfante, 1999b) of the transfinite induction theorem,

```
-subclass(P(x),x) | subclass(OMEGA,x). % ON-PC-SU
```

and the fact that the set `omega` of natural numbers is a subclass of the class `OMEGA` of ordinal numbers. This proof is reproduced in Appendix B.

The central idea underlying our approach to recursive definitions is to exploit the concept of subvariant sets (Belinfante, 2000b) to construct the class of partial solutions of recursion conditions. A set `y` is said to be *subvariant* under `x` if

$$\text{subclass}(y, \text{image}(x, y)).$$

The class of all sets subvariant under `x` is called `subvar(x)`. The proof of Lemma `TC-LEM2` uses Theorem `SBV-SC2`, which states that the sum class of the class of all subvariant sets is invariant:

```
equal(image(x,U(subvar(x))),U(subvar(x))). % SBV-SC2
```

In the general theory of subvariant sets, initial conditions in recursion equations can always be eliminated by incorporating them into the relation to which `subvar` is being applied. Recently `Otter` was used to prove two new theorems about this process:

```
equal(union(intersection(x,U(subvar(union(id(x),y))))),
      image(y,U(subvar(union(id(x),y))))),
      U(subvar(union(id(x),y)))). % SBV-U-4
```

```
equal(union(x,image(y,U(subvar(union(id(x),y))))),
      U(subvar(union(id(x),y)))). % SBV-U-5
```

The second of these new theorems enters into the proof of Lemma `TC-LEM3`.

Among the technical lemmas, Lemma `TC-LEM4` has the distinction of having the longest proof; `Otter` found a 17 step proof on level 7 in 35 seconds. By contrast,

the proofs of most of the other technical lemmas took less than 5 seconds. Despite the length of the proof for this lemma, the facts used in the proof are elementary. Although of course one needs to use the definition

```
(all x (full(x) <-> subclass(U(x),x))).           % DF-FUL
```

of the predicate `full`, no theorems about full sets are needed.

With these technical lemmas out of the way, the door to the theorems about transitive closure is thrown wide open.

Our definition `DEF-TC` of the transitive closure `tc(x)` of a class `x` is valid whether or not `x` is a set. This rather technical definition uses `subvar` to express the transitive closure as the union of partial solutions of a recursion condition. The simpler formula given in Theorem `TC-A-2` expresses the transitive closure of a set `x` as the intersection of all full sets that contain `x`. Although this formula only holds for sets, one can generalize it by writing the transitive closure of a proper class `x` as the union of the transitive closures of all subsets of `x`. This idea is expressed as the formula `tc(x) = U(image(TC,P(x)))`, and could be proved as a corollary of Theorem `TC-IM-2A`. Since the proofs of these theorems depend on the formula `U(FULL) = V`, neither one of them is suitable as a starting point. Of the theorems in the second `TC` group, the lengthiest were the 10 step proof of `TC-SC` found by `Otter` in 72 seconds, and the 11 step proof of `TC-U`, which took 28 seconds.

The definition `DEF-TCF` of the function `TC` which takes a set `x` to its transitive closure `tc(x)` can be deduced from Theorem `TC-A-2`. The vertical section equation `TC-VS` goes beyond the application formula `TC-AP` by incorporating the facts that `TC` is a function with domain `V`, and also has the advantage that `Otter` can turn this equation into a demodulator. This vertical section demodulator greatly simplifies the proofs of theorems like `TC-CO-BC`, which says the functions `BIGCUP` and `TC` commute. These demodulators are a small sample of equations about `TC` discovered using the `GOEDEL` program that have been added as rewrite rules in the current version.

5 Applications and Conclusion

For most applications of automated reasoning in modern mathematics a substantial amount of set theory is essential. Much progress has already been made toward mechanizing set theory, especially by Paulson and his coworkers (1993, 1996), the Mizar group (1999), and Megill (1997), among others. Each of these groups uses different axioms for set theory, and the methods employed differ greatly. What will eventually turn out to be the most useful approach remains to be seen.

Although the equation `U(FULL) = V` was not explicitly needed for any of the theorems that we proved about ordinal numbers, its presence as a demodulator would have helped to control combinatorial explosion. Because this demodulator was in fact not available at that time, we had to make do with a less attractive weaker demodulator which was easier to prove (Belinfante, 1999b).

The equation `U(FULL) = V` is needed to complete the proof (Belinfante, 2000b) that the weak and strong versions of the Axiom of Regularity are equivalent; for this application one needs Theorem `TC-REG-1`.

An important use of transitive closure is for constructing inner models of set theory. Recently **Otter** was used to prove theorems about the *hereditary core* of a class x , defined as the union of all full subsets of x ; that is,

$$H(x) = U(\text{intersection}(\text{FULL}, P(x))).$$

That the hereditary core $H(x)$ is the largest full subclass of the class x is obvious when x is a set, but for proper classes, the proof of this fact requires using the theorems about transitive closure. In particular, the class of hereditarily finite sets $H(\text{FINITE})$ provides a model of set theory minus the axiom of infinity.

Because of the usefulness of recursive definitions in developing arithmetic and other basic areas of applied set theory, it is worthwhile making whatever special efforts are needed to develop the appropriate definitions and basic lemmas needed for such applications. It is not essential to build explicit support for recursion into the theorem prover itself to deal with recursive definitions, but if not, one does need to develop techniques to efficiently exploit what is presently available.

For the work described here, a primary obstacle is finding succinct and useful definitions of the classes one needs. The notion of subvariance, which has previously been used to give a unified treatment of regular sets and finite sets, also is expected to enter the proofs of the recursion theorems needed to develop arithmetic. The **Otter** proof of the equation $U(\text{FULL}) = V$ points the way to such applications.

Appendix A. Theorems about transitive closure.

The following theorems about transitive closure were all proved using McCune's automated reasoning program **Otter**. Theorems flagged with an asterisk are (usually) added to the demodulator list. Gaps in the numbering of theorems usually indicate some lemma was automatically removed via demodulation and subsumption.

The theorems are organized into three groups, the first of which contains the technical lemmas.

```
% TC1.USE          2000/10/23
list(usable).
% Technical lemmas.
-subclass(x, union(composite(inverse(E), composite(x, inverse(E))), cart(V, y))) |
  subclass(image(x, singleton(z)), union(U(image(x, z)), y)).           % TC-LEM1A
-member(x, V) | -subclass(y, union(composite(inverse(E), composite(y, inverse(E))),
  cart(V, x))) | subclass(P(D(VERTSECT(y))), D(VERTSECT(y))).       % TC-LEM1B
-member(x, V) | -subclass(y, union(composite(inverse(E), composite(y, inverse(E))),
  cart(V, x))) | member(image(y, omega), V).                          % TC-LEM1C
-subclass(x, union(cross(E, inverse(E)), id(cart(V, y)))) | subclass(U(subvar(x)),
  union(composite(inverse(E), composite(U(subvar(x)), inverse(E))), cart(V, y))). % TC-LEM-2
equal(union(cart(V, x), composite(inverse(E),
  composite(U(subvar(union(cross(E, inverse(E)), id(cart(V, x))))), inverse(E))),
  U(subvar(union(cross(E, inverse(E)), id(cart(V, x)))))), %*TC-LEM-3
-full(x) | subclass(P(complement(image(inverse(U(subvar(union(cross(E, inverse(E)),
  id(cart(V, x)))))), complement(x))),
  complement(image(inverse(U(subvar(union(cross(E, inverse(E)),
  id(cart(V, x)))))), complement(x)))). % TC-LEM-4
```

The second group of theorems is about $tc(x)$, and some applications. The proof of $U(FULL) = V$ is in this group. Theorem TC-REG-1 is needed to complete the proof that the weak and strong versions of the axiom of regularity are equivalent (Belinfante 2000b). The last two theorems characterize the transitive closure of a set as a least fixed point.

```

% TC2.USE                2000/10/26
%
% Definition of transitive closure tc(x).
equal(image(U(subvar(union(cross(E,inverse(E)),id(cart(V,x))))),omega),tc(x)).    %*DEF-TC
% Theorems about transitive closure.
-member(x,V) | member(tc(x),V).                                                % TC-V
equal(union(x,U(tc(x))),tc(x)).                                                %*TC-EQ
subclass(x,tc(x)).                                                              % TC-SU-1
-subclass(x,y) | subclass(tc(x),tc(y)).                                        % TC-SU-2
full(tc(x)).                                                                    % TC-FUL-1
member(tc(singleton(x)),FULL).                                                 % TC-SS-1
%
% Two applications of transitive closure.
equal(U(FULL),V).                                                              %*TC-FUL-2
-subclass(P(x),x) | subclass(REGULAR,x).                                       % TC-REG-1
-full(x) | -subclass(intersection(x,P(y)),y) |
  subclass(intersection(REGULAR,x),y).                                         % TC-REG-2
%
% More about transitive closure
-full(x) | equal(tc(x),x).                                                     % TC-FUL-3
equal(tc(tc(x)),tc(x)).                                                        %*TC-TC
-full(x) | -subclass(y,x) | subclass(tc(y),x).                                % TC-FUL-4
-equal(union(x,U(y)),y) | subclass(tc(x),y).                                  % TC-FUL-5
-member(x,y) | subclass(tc(x),tc(y)).                                          % TC-SU-3
-member(x,tc(y)) | -member(y,tc(z)) | member(x,tc(z)).                       % TC-SU-4
subclass(tc(x),A(intersection(FULL,image(S,singleton(x))))).                  % TC-A-1
-member(x,V) | equal(A(intersection(FULL,image(S,singleton(x))))),tc(x)).     % TC-A-2
equal(tc(U(x)),U(tc(x))).                                                       %*TC-SC
-member(x,V) | equal(U(tc(singleton(x))),tc(x)).                               % TC-SS-2
-member(x,V) | equal(tc(singleton(x)),union(singleton(x),tc(x))).             % TC-SS-3
equal(tc(P(x)),union(tc(x),P(x))).                                             %*TC-PC-1A
equal(tc(union(x,y)),union(tc(x),tc(y))).                                     %*TC-U
-member(x,V) | member(tc(x),fix(composite(ADJOIN(x),BIGCUP))).                % TC-FP-1
-member(x,V) | equal(A(fix(composite(ADJOIN(x),BIGCUP))),tc(x)).              % TC-FP-2

```

The third group is concerned with the function TC, and with the proof that $composite(inverse(E),TC)$ is the smallest transitive relation which contains the inverse of the membership relation. The longest proofs found were the 28 step proof of TC-TRV, obtained on level 13 in 65 seconds, and the 19 step proof of TR-IN-E found in 23 seconds.

```

% TC3.USE                2000/10/26
% Definition of the transitive closure operation TC
equal(VERSECT(complement(composite(complement(inverse(E)),
  composite(id(FULL),S))),TC).                                                %*DEF-TCF
%
% Theorems about the transitive closure operation.
FUNCTION(TC).                                                                    % TC-FU
equal(D(TC),V).                                                                  %*TC-DO
thin(complement(composite(complement(inverse(E)),composite(id(FULL),S)))).    % TC-TH
-member(x,V) | equal(apply(TC,x),tc(x)).                                        % TC-AP
equal(image(TC,singleton(x)),singleton(tc(x))).                                %*TC-VS
equal(composite(TC,BIGCUP),composite(BIGCUP,TC)).                             %*TC-CO-BC
-member(x,V) | member(pair(x,tc(x)),TC).                                       % TC-OP1
-member(pair(x,y),TC) | equal(tc(x),y).                                       % TC-OP2

```

```

subclass(TC,S). % TC-SR
thin(inverse(TC)). % TC-IN-TH
-member(tc(x),y) | member(x,image(inverse(TC),y)). % TC-IMIN1
-member(x,image(inverse(TC),y)) | member(tc(x),y). % TC-IMIN2
full(image(inverse(TC),P(x))). % TC-IMIN3
subclass(inverse(E),composite(inverse(E),TC)). % TC-IN-E
TRANSITIVE(composite(inverse(E),TC)). % TC-TRV
full(U(tc(x))). % TC-IM1A'
equal(U(image(TC,x)),U(tc(x))). %*TC-IM-2A
-TRANSITIVE(x) | -subclass(inverse(E),x) | subclass(composite(inverse(E),TC),x). % TC-CLOS
equal(R(TC),FULL). %*TC-RA2
equal(composite(TC,inverse(TC)),id(FULL)). %*TC-CO-IN
equal(fix(TC),FULL). %*TC-FP2
equal(composite(TC,E),composite(id(FULL),E)). %*TC-CO-E
equal(U(image(inverse(TC),x)),U(intersection(FULL,x))). %*TC-IMIN4
end_of_list.

```

Appendix B. Otter's proof of Lemma TC-LEM1C.

Transfinite induction enters into the proofs of the transitive closure theorems only via the technical lemma TC-LEM1C. The short proof found by Otter is reproduced below. The clauses numbered 53 to 3451 were selected by Otter from the 3451 available clauses of previously proven theorems provided in the input file. The clause 3451 is Lemma TC-LEM1B. By Skolemizing the negation of the statement of the lemma, Otter automatically produced Clauses 3452 through 3454. Clause 4347 is the sole demodulator that Otter selected for this proof. The remaining lines represent the actual proof, each of which uses unit resolution, except of course the final clause, for which binary resolution is always used.

----- SUMMARY FOR THEOREM TC-LEM1C IN DIRECTORY Q:\tc\2 -----

% TC-LEM1C.IN 2000 October 16

```

formula_list(sos).
% negation of Theorem TC-LEM1C
-(all x y ((member(x,V) & subclass(y,union(composite(inverse(E),
    composite(y,inverse(E))),cart(V,x)))) -> member(image(y,omega),V))).
end_of_list.

```

----- Otter 3.0.4, August 1995 -----

The job was started by Belinfante on Gateway P500, Mon Oct 23 15:10:41 2000

Length of proof is 5. Level of proof is 3.

----- PROOF -----

```

53 [] member(omega,V).
77 [] -subclass(x,y) | -subclass(y,z) | subclass(x,z).
654 [] -member(x,V) | -subclass(x,y) | member(x,P(y)).
2625 [] -member(x,D(IMAGE(y))) | member(image(y,x),V).
3274 [] -subclass(P(x),x) | subclass(OMEGA,x).
3294 [] subclass(omega,OMEGA).
3451 [] -member(x,V) | -subclass(y,union(composite(inverse(E),
    composite(y,inverse(E))),cart(V,x))) | subclass(P(D(VERTSECT(y))),D(VERTSECT(y))).
3452 [] member($c2,V).
3453 [] subclass($c1,union(composite(inverse(E),composite($c1,inverse(E))),cart(V,$c2))).
3454 [] -member(image($c1,omega),V).
4347 [] equal(D(IMAGE(x)),P(D(VERTSECT(x)))).
4570 [ur,3453,3451,3452] subclass(P(D(VERTSECT($c1))),D(VERTSECT($c1))).
4571 [ur,3454,2625,demod,4347] -member(omega,P(D(VERTSECT($c1)))).

```

```
4575 [ur,4570,3274] subclass(OMEGA,D(VERTSECT($c1))).
4578 [ur,4571,654,53] -subclass(omega,D(VERTSECT($c1))).
4587 [ur,4575,77,3294] subclass(omega,D(VERTSECT($c1))).
4588 [binary,4587.1,4578.1] $F.
```

----- end of proof -----

```
clauses generated    19287
Kbytes malloced      3225
user CPU time        4.23 seconds
```

References

- Belinfante, J. G. F., On a Modification of Gödel's Algorithm for Class Formation, Association for Automated Reasoning News Letter, No. 34 (1996) pp. 10–15.
- Belinfante, J. G. F., On Quaife's Development of Class Theory, Association for Automated Reasoning Newsletter, No. 37 (1997) pp. 5–9.
- Belinfante, J. G. F., Computer Proofs in Gödel's Class Theory with Equational Definitions for Composite and Cross, Journal of Automated Reasoning, vol. 22 (1999) pp. 311–339.
- Belinfante, J. G. F., On Computer-Assisted Proofs in Ordinal Number Theory, Journal of Automated Reasoning, vol. 22 (1999), pp. 341–378.
- Belinfante, J. G. F., Gödel's Algorithm for Class Formation, in: Automated Deduction–CADE-17, edited by D. McAllester, June 2000, Lecture Notes in Artificial Intelligence, vol. 1831, pp. 132–147, Springer-Verlag, Berlin. (ISBN 3-540-67664-3)
- Belinfante, J. G. F., The Unifying Concept of Subvariance, in FTP 2000, Third International Workshop on First-Order Theorem Proving, St. Andrews, Scotland, July 2000, edited by P. Baumgartner and H. Zhang, pp. 56–67, Fachberichte Informatik, Universität Koblenz-Landau
- Bernays, P., Axiomatic Set Theory, North Holland Publishing Co., Amsterdam. First edition: 1958. Second edition: 1968. Republished in 1991 by Dover Publications, New York.
- Boyer, R., Lusk, E., McCune, W., Overbeek, R., Stickel M. and Wos, L., Set Theory in First Order Logic: Clauses for Gödel's Axioms, Journal of Automated Reasoning, volume 2 (1986), pages 287–327.
- Formisano, A. and Omodeo, E. G., An Equational Re-Engineering of Set Theories, presented at the FTP'98 International Workshop on First Order Theorem Proving (November 23–25, 1998).
- Gödel, K., The Consistency of the Axiom of Choice and of the Generalized Continuum Hypothesis with the Axioms of Set Theory, Princeton University Press, Princeton, 1940.
- McCune, W. W., Otter 3.0 Reference Manual and Guide, Argonne National Laboratory Report ANL-94/6, Argonne National Laboratory, Argonne, IL, January 1994.
- Megill, N. D., Metamath: A Computer Language for Pure Mathematics, 1997.
- Noël, P. A. J., Experimenting with Isabelle in ZF set theory, Journal of Automated Reasoning, vol. 10 (1993), pp. 15–58.
- Paulson, L. C., and Grabczewski, K., Mechanizing Set Theory, Journal of Automated Reasoning, vol. 17, pp. 291–323 (1996).
- Quaife, A., Automated Deduction in von Neumann-Bernays-Gödel Set Theory, Journal of Automated Reasoning, vol. 8 (1992), pp. 91–147.
- Quaife, A., Automated Development of Fundamental Mathematical Theories, Ph.D. thesis, Univ. of California at Berkeley, Kluwer Acad. Publishers, Dordrecht, 1992.
- Rubin, J. E., Set Theory for the Mathematician, Holden-Day, San Francisco, 1967.
- Rudnicki, P., and Trybulec, A., On equivalents of well-foundedness, Journal of Automated Reasoning, vol. 23 (1999), pp. 197–234.
- Tarski, A., and Givant, S., A Formalization of Set Theory without Variables, American Mathematical Society Colloquium Publications, volume 41, Providence, Rhode Island, 1987.
- Wos, L., Automated Reasoning: 33 Basic Research Problems, Prentice Hall, Englewood Cliffs, NJ, 1988.
- Wos, L., The problem of finding an inference rule for set theory, Journal of Automated Reasoning, vol. 5 (1989), pp. 93–95.
- Wos, L., Overbeek, R., Lusk, E. and Boyle, J., Automated Reasoning: Introduction and Applications, Second Edition, McGraw Hill, New York, 1992.
- Wolfram, S., The MathematicaTM Book, Wolfram Media Inc., Champaign, Illinois, 1996.