Journal of Automated Reasoning **8**: 91–147, 1992.
© 1992 *Kluwer Academic Publishers. Printed in the Netherlands.*

Automated Deduction in von Neumann–Bernays–Gödel Set Theory

ART QUAIFE
Trans Time, Inc., 10208 Pearmain St., Oakland, CA 94603, U.S.A.

(Received: 31 October 1989, accepted 6 April 1990)

Abstract. I present a new clausal version of NGB set theory, and compare my version with that first given by Boyer *et al.* [4]. A complete set of reductions for Boolean rings is given, derived from those of Hsiang [7]. I list over 400 theorems proved semiautomatically in elementary set theory, and supply the proofs of several of these, including Cantor's theorem. I present a semiautomated proof that the composition of homomorphisms is a homomorphism, thus solving a challenge problem given in [4]. Using the clauses and heuristics presented, there is no apparent obstacle to the semiautomated development of set theory through considerably more difficult theorems.

Key words. Set theory, automated reasoning, mechanical theorem proving, reductions, heuristics.

1. Introduction

Since virtually all extant mathematics can be formulated in the language of set theory, this theory could be regarded as the ultimate proving ground for automated theorem-proving programs. However, very simple theorems in set theory have proved difficult for past resolution theorem provers. Winker and Wos [20], in connection with trying to prove that union distributes over intersection, state: "Many are acquainted with the disappointing tediousness with which such problems are solved with standard theorem-proving approaches." Lusk and Overbeek [10] state "... there are a variety of quite simple problems that have stymied resolution-based programs for years. [Proving] the union of sets is commutative has proved challenging for [resolution] programs." I address these particular difficulties in Section 5 of this paper.

Boyer *et al.* [4] have performed a useful service by providing a clausal version of the von Neumann–Bernays–Gödel (NBG) version of set theory, suitable as input to resolution theorem provers. They follow the treatment of Gödel [6], but have gone beyond simply using standard conversion procedures to translate first-order NBG set theory into clausal form. In several instances where Gödel asserts the existence of a set with certain properties, they have uniquely determined the set and given it a name. In addition to the axioms, they have provided the clausal form for a number of definitions and theorems from set theory, abstract algebra, and number theory.

Boyer *et al.* state that "Anyone attempting to submit the clauses given here to an automated theorem prover will quickly confront many fundamental issues in theorem proving. This set of clauses is difficult to work with for several reasons. [Four reasons are then enumeratured.]" However, their conversion to clausal form can be improved in quite a few ways, which I describe in Section 3. I present a new conversion in Section 4. The resulting clauses are much more amenable to automated development, and I list over 400 theorems in elementary set theory proved with their use.

Section 6 contains a complete set of reductions for Boolean rings, revised from those given by Hsiang [7] so they will work in a reasoning system not embodying associative-commutative unification. Section 7 describes the relationship between Skolem functors and the Axiom of Choice.

The theorems I have proved using this set of clauses are introduced in Section 8. Sections 9, 10 and 11 discuss the heuristics and option settings used to obtain proofs of these theorems. Section 12 contains a machine proof of Cantor's theorem.

Boyer *et al.* [4] challenge readers to find an automated proof that the composition of homomorphisms is a homomorphism, which is also Test Problem 15 in Wos [22]. They then provide a sequence of 27 lemmas that lead to its proof, along with the proofs of these lemmas, which they obtained by hand and verified by machine. Their mechanical proof checker is guided by user instructions such as "compute all the binary resolvents of clause 42 and clause 76". They assert, "It is not our intention to suggest that any existing resolution-based theorem prover ought to be able to attack the example challenge problems successfully. Indeed, we suspect that all of the challenge problems are probably beyond that ability of any current known resolution-based system."

This assertion notwithstanding, I have obtained semiautomated proofs (see Section 11) of all these lemmas, and of the theorem itself. These results are presented in Section 13.

Possible improvements to the unification algorithm appropriate to NBG set theory are discussed in Section 14.

For readers who are unfamiliar with resolution theorem proving in general, and OTTER in particular, Appendix 1 provides a brief introduction. This appendix explains such concepts as clauses, substitutions, unification, binary resolution, hyperresolution, UR-resolution, paramodulation, demodulation, set-of-support strategy, subsumption, weighting, and lexical ordering.

Appendix 2 contains a list of more than 400 theorems in elementary set theory proved using the clauses and methods presented in the paper.

Thus the main contributions of this study are:

A. I provide a clausal version of NBG set theory that is amenable to semiautomated development using a resolution theorem prover.
B. I provide a number of heuristics and proof procedures that have proved effective in carrying out this development.
C. I demonstrate the value of these clauses and proof procedures by presenting theorems and proofs obtained by their use.

2. Notation

Names beginning with small '*u*' through '*z*' are variables. All other names are either individual constants, function symbols, or relation symbols, depending upon the context. Formulas containing the sequent sign '$\rightarrow$' are clauses. All other formulas are formulas of first-order logic.

Above I used 'function symbol' in the usual sense of first-order logic. But our subject matter here is set theory, and among the objects most studied are functions as classes of ordered pairs. Thus, for example, '*I*' is an individual constant that names the identity function. To avoid possible confusion, from now on I will use 'functor' rather than 'function symbol'. I further discuss the relation between terms and functions in Section 3.6.

I present clauses essentially as I load them into OTTER version 1.01, written by William McCune of Argonne National Laboratory [12]. OTTER is a fast resolution theorem prover, to which I have added the capabilities of using sequent notation, and of accepting one-character infix operators. To improve readability, I replace some ASCII symbols used in OTTER by standard set-theoretic notation.[1]

I present clauses fully parenthesized, whereas I often omit parentheses from first-order formulas when no confusion can result. I write clausal equalities as they will be ordered by OTTER. This explains why in definitions the defined term is on the right side of the equality - it is of lower weight than the defining term.

3. Simplications

In this section I describe the modifications I made to Gödel's axioms while converting them to clausal from. Since an extended development of virtually the whole of mathematics can be based on these axioms, efforts toward expressing them as simply as possible should be amply rewarded. Considerations of machine efficiency will be important in my conversion, and thus the clauses I supply do not result from a direct clausification of Gödel's axioms. Rather, I believe that with proper definitions they are provably equivalent to this. I compare my conversion to that of Boyer *et al.*

3.1. SETHOOD

Gödel's axioms contain a unary relation $M(x)$, interpreted as 'x is a set'. He defines the universal class V, and has as a theorem

$$x \in V \Leftrightarrow M(x).$$

[1]Symbols appearing in this paper are approximated in OTTER as follows:

$\in$	e	‘	`	$\rightarrow$	– >	$(x \times y)$	X(x, y)
$\subseteq$	<	“	”	$\{x, y\}$	$(x\ ;\ y)$	ω	omega
$\cup$	+	$\circ$	¯	$\{x\}$	ss(x)		
$\cap$	*	+	&	$\langle x, y\rangle$	$[x, y]$		

Thus as our first simplification, we will use this equivalence to eliminate the unary relation M in favor of the individual constant V, making this replacement throughout his axioms. Without this elimination, there is frequent necessity to make the equivalence deduction in one direction or the other.

3.2. EQUALITY

The equality axiom $\rightarrow (x = x)$ of Boyer *et al.* is not needed as an axiom, since it follows from the Axiom of Extensionality. I include it in the axiom and theorem list loaded to OTTER as Theorem (EQ1) (see Appendix 2).

Their version of the Axiom of Extensionality contains a Skolem functor, and does not work well with hyperresolution or UR-resolution. It does not permit a natural way to split up a proof that $\rightarrow (x = y)$. It is preferable first to define the subclass relation $\subseteq$, which of course will be needed anyway, and express extensionality as

$$x = y \Leftrightarrow x \subseteq y \;\&\; y \subseteq x.$$

This requires no new Skolem functors, and provides the natural breakdown: to show two classes are equal, show that each is a subclass of the other.

I further discuss proving equality of classes in Section 5.

3.3. INTRODUCTION OF ORDERED PAIRS

I have completely revised their treatment of the first and second components of ordered pairs, and have eliminated their ordered pair predicate *OPP*.

I first prove the following theorem:

$$\begin{aligned}&\forall x\, \exists u\, \exists v((\langle u, v\rangle \in (V \times V) \;\&\; x = \langle u, v\rangle)\\ &\quad \vee (\neg \exists y\, \exists z(\langle y, z\rangle \in (V \times V) \;\&\; x = \langle y, z\rangle) \;\&\; u = x \;\&\; v = x)).\end{aligned}$$

Skolemizing this theorem produces two functors, '*1st*' and '*2nd*', which are similar to the '*first*' and '*second*' functors that they introduce by separate and unneeded axioms. A difference is that in the case x is not an ordered pair, I require $1st(x) = x$ and $2nd(x) = x$, whereas Boyer *et al.* make their default values 0. Note that we can use the Axiom of Regularity to show that $\langle x, y\rangle \neq x$ for sets x and y. Knowing that $1st(x) = 0$ does not tell one whether x is indeed an ordered pair of sets, whereas from $1st(x) = x$ one can immediately conclude that x is *not* an ordered pair of sets.

I also prove the characteristic uniqueness theorems:

$$(\langle u, v\rangle \in (V \times V)) \rightarrow (1st(\langle u, v\rangle) = u).$$

$$(\langle u, v\rangle \in (V \times V)) \rightarrow (2nd(\langle u, v\rangle) = v).$$

I also prove uniqueness in the case that the argument is *not* an ordered pair of sets.

These theorems eliminate the need for their Axioms of First and Second, and the four Skolem functors in them. I also eliminate their definiton of the ordered pair predicate $OPP(x)$, eliminating two Skolem functions and four clauses. Instead of $OPP(x)$ we will use $(x \in (V \times V))$ or, rarely, $(\langle 1st(x), 2nd(x)\rangle = x)$.

3.4 USE OF ORDERED PAIRS

The clauses in Boyer *et al.* contain many instances of the sethood predicate $(x \in V)$. Some clauses have four instances of it! Using their clauses I have found that whenever a newly deduced unit $\rightarrow (t \in V)$ becomes the given clause, the system bogs down to a crawl while hyperresolution or UR-resolution tests the huge number of unifying matches – especially in clauses with multiple appearances.

There is a fairly simple solution to this problem. Rather than using $(x \in V)$, $(y \in V) \rightarrow$ in the hypothesis of a clause, instead use $(\langle x, y\rangle \in (V \times V)) \rightarrow$. Not only does this reduce the number of literals in the clauses, but we also must use the clause

$$(x \in V), (y \in V) \rightarrow (\langle x, y\rangle \in (V \times V))$$

to make the further deduction that some pair is ordered. This delays the deduction of new ordered pair literals, improves the combinatorics substantially, and largely eliminates this problem. If we are ever confronted with a glut of deduced clauses $\rightarrow(\langle t_1, t_2\rangle \in (V \times V))$, we could selectively block their deduction (via the displayed clause) by using weights.

There are two different forms that can be used for clauses containing ordered pairs. To illustrate, Boyer *et al.* give Axiom B-1, which defines the elementhood relation, as

$$(z \in E) \Leftrightarrow (z \in (V \times V)) \;\&\; (1st(z) \in 2nd(z)).$$

The direct clausification produces

$$(z \in E) \rightarrow (z \in (V \times V)).$$

$$(z \in E) \rightarrow (1st(z) \in 2nd(z)).$$

$$(z \in (V \times V)), (1st(z) \in 2nd(z)) \rightarrow (z \in E).$$

Or using simple theorems about *1st* and *2nd*, we can rewrite these as

$$(z \in E) \rightarrow (z \in (V \times V)) \quad \{\text{or} \rightarrow (E \subseteq (V \times V))\}.$$

$$(\langle x, y\rangle \in E) \rightarrow (x \in y).$$

$$(\langle x, y\rangle \in (V \times V)), (x \in y) \rightarrow (\langle x, y\rangle \in E).$$

This second set of clauses also comes directly from the version of the class existence theorem given below in Section 3.6. Boyer *et al.* use the first form. I always use the second form. I believe these latter clauses are more natural, and appear to be of the form most often directly useful.

3.5. CONSTRUCTOR AXIOMS

Gödel's Axiom B-6 (inverse) is dependent. I rearranged the order of axiom introduction so that we can prove B-6 as a theorem. This treatment is inspired by Mendelson [13].

Boyer *et al.* use a version of Axiom B-5 that gives the full Cartesian product $(x \times y)$, whereas Gödel's version only gives the existence of $(V \times y)$. I will follow Boyer *et al.*, and furthermore list Axiom B-5′ as the first of the Group B axioms, which permits a more convenient expression of the remaining Group B axioms. The alternative is to use more convoluted axioms; then after proving the existence of the inverse of a relation define $(x \times y) = (V \times y) \cap \mathit{inverse}((V \times x))$, and finally use this definiton to back-simplify the axioms.

For reasons given at the beginning of Section 3.4, we wish to use Axiom B-5′a in the unrelativized form given. However this requires that no ordered pair in which one of the arguments is a proper class can be equal to an ordered pair of sets. The usual definition of the ordered pair as $\langle x, y\rangle = \{\{x\}, \{x, y\}\}$ fails this test, for using it we have

$$\langle 0, 0\rangle = \{\{0\}, \{0, 0\}\} = \{\{0\}, \{0\}\} = \{\{0\}, \{0, V\}\} = \langle 0, V\rangle.$$

Then using B-5′a, we would obtain the false conclusion

$$\langle 0, 0\rangle \in (V \times V) \Rightarrow \langle 0, V\rangle \in (V \times V) \Rightarrow V \in V.$$

Thus I will instead define $\langle x, y\rangle = \{\{x\}, \{x, \{y\}\}\}$.

3.6. CLASS EXISTENCE THEOREM

Boyer *et al.* chose the von Neumann–Bernays–Gödel version of set theory, rather than Zermelo–Fraenkel (ZF), because the *finite* number of axioms in NBG can all be input to a computer. But note that very early in the development of NBG one proves the class existence theorem:

Let $\phi(x_1, \ldots, x_n, y_1, \ldots, y_m)$ be a formula in the primitive notation of NBG whose free variables are among $x_1, \ldots, x_n, y_1, \ldots, y_m$, and in which all quantifiers are relativized to V (ϕ is *predicative*). Then

$$\exists! z((z \subseteq V^n) \mathbin{\&} \forall x_1 \ldots \forall x_n((\langle x_1, \ldots, x_n\rangle \in z) \Leftrightarrow ((\langle x_1, \ldots, x_n\rangle \in V^n) \mathbin{\&} \phi(x_1, \ldots, x_n, y_1, \ldots, y_m)))).$$

This metaschema in NBG is analogous to the Axiom Schema of Subsets in ZF, and the two metaschemata are used approximately as often in the respective developments.

We can expand the primitive notation of NBG by permitting the introduction of new functors whenever the usual existence and uniqueness theorems are proved. Such functors are, in principle, eliminable. There is a similar metatheorem that I will call the function existence theorem: suppose that ϕ is predicative and τ is a functor such that

$$\forall x_1 \ldots \forall x_n((\langle x_1, \ldots, x_n\rangle \in V^n) \Rightarrow ((\tau(x_1, \ldots, x_n) \in V) \mathbin{\&} \forall y((y \in \tau(x_1, \ldots, x_n)) \Leftrightarrow \phi(x_1, \ldots, x_n, y)))).$$

Then

$$\exists! xf((xf \subseteq V^{n+1}) \mathbin{\&} \forall x_1 \ldots \forall x_n \forall z((\langle x_1, \ldots, x_n, z\rangle \in xf) \Leftrightarrow ((\langle x_1, \ldots, x_n\rangle \in V^n) \mathbin{\&} \tau(x_1, \ldots, x_n) = z)).$$

Thus under appropriate conditions, formulas may be reified into classes and terms may be reified into functions.

There are are least three approaches we can take toward using the class existence theorem:

1. Make use of this theorem to define manually new classes as needed, adding axioms that are called definitions. Boyer *et al.* adopted this approach. Of course, we must define such classes before any proof run that needs them.
2. Automate the introduction of newly defined classes, justified by the class existence theorem, as part of a metalevel control mechanism.

In these two approaches, the practical advantage of the finite axiomatizability of NBG is largely lost and illusory. It has not done away with the need for the automated reasoning system to use infinite schemata of axioms/theorems, either clumsily by hand or else mechanically. The second approach may be worth pursuing, but such automation of the metatheory is not available in OTTER version 1.01.

In contrast to the above two approaches, the approach I adopt makes essential use of the finite axiomatizability of NBG. In particular, note that most of the definitions via the class existence theorem in Boyer *et al.* contain quantifiers, and thus produce Skolem functors when clausified. But use of Axiom B-4 (domain) allows us to mirror such definitions without using quantifiers, thus eliminating the Skolem functors. Thus the approach I have adopted is:

3. Generally avoid using the class existence theorem, especially when its use would introduce Skolem functors. Instead, whenever one needs to define a new class, give an explicit definition which mirrors the construction techniques used in the proof of the theorem. For example, rather than Boyer *et al.*'s definition of the sum class $U(x)$ by

   $$(z \in U(x) \Leftrightarrow (z \in V \mathbin{\&} \exists y((y \in V) \mathbin{\&} (z \in y) \mathbin{\&} (y \in x))),$$

   we can use either of the explicit definitions

   $$\begin{aligned} U(x) &= (\mathit{inverse}(E) \text{``} x) \\ &= D(\mathit{restrict}(E, V, x)) \end{aligned}$$

   and eliminate a Skolem functor.

The construction techniques used in the proof of the class existence theorem can be improved upon to produce simpler definitions. In particular, we can often eliminate an existential quantifier by using

$$\exists z((\langle x, z\rangle \in u) \;\&\; (\langle z, y\rangle \in v)) \;\Leftrightarrow\; (\langle x, y\rangle \in (v \circ u)).$$

3.7. CONSTRUCTORS VERSUS SKOLEM FUNCTORS

As a further example, Boyer *et al.* define the *image* functor by the first-order equivalence (changed to my notation):

$$\forall y\, \forall z\, \forall xf(y \in (xf\text{“}z) \Leftrightarrow$$

$$y \in V \;\&\; \exists u(u \in V \;\&\; u \in (V \times V) \;\&\; u \in xf \;\&\; \mathit{1st}(u) \in z \;\&\; \mathit{2nd}(u) = y)).$$

The formula on the right contains a quantifier, which produces a Skolem functor upon clausification. Such quantifiers in the defining formulas they use produce a glut of Skolem functors, identified only as $f1, \ldots, f59$.

William of Ockham has warned us against multiplying Skolem functors needlessly. In the first place, it is generally preferable to construct and uniquely determine an object than simply to assert its existence. In the second place, these Skolem functors are not independent, but with more careful work can be interdefined. Thirdly, Skolem functors may *not* appear in formulas ϕ used to define further classes via the class existence theorem (see Section 7). Finally, a user may be excused if upon seeing a formula containing $(f31(x) \in f43(y))$, he wonders what the formula is about. I will instead rely heavily upon the Group B axioms, which provide the means for constructing new classes from old ones by explicit equality definitions.

The Group B axioms produce constructor functors under clausification, which may be defined uniquely (as Boyer *et al.* have done). I have also modified the Axiom of Infinity so that the infinite set is uniquely determined to be ω. There are three other axioms that unavoidably produce Skolem functors or individual constants: the Axioms of Extensionality, Regularity, and Choice. Extensionality is, of course, fundamental, and relies on the Skolem functor '*notsub*' that comes from the definition of subclass. (If x is not a subset of y, then *notsub*(x, y) witnesses the fact by belonging to x but not y.) Of the 59 Skolem functors appearing in the clauses of Boyer *et al.*, about 56 of them can be explicitly defined in terms of the Skolem functor '*notsub*' and the constructor functors of Group B. I have done so, except for the two Skolem functors in the definition of *HOM* (homomorphism), whose elimination appears to be too tedious. I have also given names to the few Skolem functors that remain after these simplifications. The names are related to their purpose, and should make the axioms more intelligible to the user.

If a defined class can be uniquely determined, *notsub* is not needed in the definition. As an example of where it is needed, see the definitions of *dom* and *ran* in Section 4, which can be used to replace quite a few of the Skolem functors in Boyer *et al.*

Continuing with the example of the *image* functor, the definition given above clausifies to

$$(y \in (xr \text{``} x)) \rightarrow (f22(y, x, xr) \in (V \times V)).$$

$$(y \in (xr \text{``} x)) \rightarrow (f22(y, x, xr) \in xr).$$

$$(y \in (xr \text{``} x)) \rightarrow 1st(f22(y, x, xr)) \in x).$$

$$(y \in (xr \text{``} x)) \rightarrow (2nd(f22(y, x, xr)) = y).$$

$$(u \in xr), (u \in (V \times V)), (1st(u) \in z), (2nd(u) = y), (y \in V) \rightarrow (y \in (xr \text{``} z)).$$

I instead use a one-line definition of this functor:

$$\rightarrow (R(restrict(xr, x, V)) = (xr \text{``} x)).$$

Normally when I give an explicit construction of a class, I also prove the theorems that give the simplest membership conditions for the class. Here, they are

$$(y \in (xr \text{``} x)) \rightarrow (\langle dom(xr, x, y), y\rangle \in (V \times V)).$$

$$(y \in (xr \text{``} x)) \rightarrow (\langle dom(xr, x, y), y\rangle \in xr).$$

$$(y \in (xr \text{``} x)) \rightarrow (dom(xr, x, y) \in x).$$

$$(\langle x, y\rangle \in xr), (\langle x, y\rangle \in (V \times V)), (x \in z) \rightarrow (y \in (xr \text{``} z)).$$

Note in particular that we have replaced the Skolem functor '*f22*' by the functor '*dom*'. In Section 4 *dom*(*xr*, *x*, *y*) is defined to be a preimage of *y* in *x* under *xr*, and numerous properties of *dom* are proved in Appendix 2 beginning with the section RANGE. We also see how replacing *u* by $\langle x, y\rangle$ has eliminated a clause and simplified the last clause.

Sometimes explicit definitions are not illuminating. For example, we define the successor functor by

$$succ(x) = x \cup \{x\},$$

and we can explicitly define the corresponding function (relation) by

$$SUCC = (V \times V) \cap \sim(((E \circ \sim(inverse((E \cup I))))$$
$$\cup (\sim(E) \circ inverse((E \cup I))))).$$

It would be tedious to prove the desired membership properties of this relation, and so we instead use the definition given by the class existence theorem:

$$SUCC \subseteq (V \times V),$$

$$(\langle x, y\rangle \in SUCC) \Leftrightarrow (\langle x, y\rangle \in (V \times V)) \;\&\; (succ(x) = y),$$

which in this case does not produce any Skolem functors. (We could instead use the function existence theorem after proving (SS1) and the corollary to (SC5) in Appendix 2.)

The net effect of these simplifications on the set theory clauses of their GROUP 1 and GROUP 2 is a reduction from 33 to 5 Skolem functors, and from 142 to 90 clauses.

4. Clauses for Axioms and Definitions

For each axiom and definition, I first present its version in first-order logic, then its clausal version. Referenced theorems may be found in Appendix 2.

GROUP 1: AXIOMS AND BASIC DEFINITIONS

Axiom A-1: Sets are classes (omitted because all objects are classes).

Definition of $\subseteq$ (subclass).
$\forall x\, \forall y((x \subseteq y) \iff \forall u((u \in x) \Rightarrow (u \in y)))$.

$(x \subseteq y), (u \in x) \to (u \in y)$.
$\to (notsub(x, y) \in x), (x \subseteq y)$.
$(notsub(x, y) \in y) \to (x \subseteq y)$.

Axiom A-2: Elements of classes are sets.
$\forall x\, (x \subseteq V)$.

$\to (x \subseteq V)$.

Axiom A-3: Extensionality.
$\forall x\, \forall y((x = y) \iff (x \subseteq y)\ \&\ (y \subseteq x))$.

$(x = y) \to (x \subseteq y)$.
$(x = y) \to (y \subseteq x)$.
$(x \subseteq y), (y \subseteq x) \to (x = y)$.

Axiom A-4: Existence of unordered pair.
$\forall u\, \forall x\, \forall y((u \in \{x, y\}) \iff (u \in V)\ \&\ (u = x \lor u = y))$.
$\forall x\, \forall y(\{x, y\} \in V)$.

$(u \in \{x, y\}) \to (u = x), (u = y)$.
$(x \in V) \to (x \in \{x, y\})$.
$(y \in V) \to (y \in \{x, y\})$.
$\to (\{x, y\} \in V)$.

Definition of singleton set.
$\forall x(\{x\} = \{x, x\})$.

$\to (\{x, x\} = \{x\})$.

Theorem (SS6) in Appendix 2 introduces *memb*.

Definition of ordered pair.
$\forall x\, \forall y(<x, y> = \{\{x\}, \{x, \{y\}\}\})$.

$\to (\{\{x\}, \{x, \{y\}\}\} = <x, y>)$.

Axiom B-5'a: Cartesian product.
$\forall u\, \forall v\, \forall x\, \forall y((<u, v> \in (x \times y)) \iff (u \in x)\ \&\ (v \in y))$.

$(<u, v> \in (x \times y)) \to (u \in x)$.
$(<u, v> \in (x \times y)) \to (v \in y)$.
$(u \in x), (v \in y) \to (<u, v> \in (x \times y))$.

See Theorem (OP6) for *1st* and *2nd*.

Axiom B-5'b: Cartesian product.
$\forall z(z \in (x \times y) \Rightarrow (z = <1st(z), 2nd(z)>)$.

$(z \in (x \times y)) \rightarrow (z = <1st(z), 2nd(z)>)$.

Axiom B-1: *E* (elementhood relation).
$(E \subseteq (V \times V))$.
$\forall x\ \forall y((<x, y> \in E) \Longleftrightarrow (<x, y> \in (V \times V)) \ \&\ (x \in y))$.

$\rightarrow (E \subseteq (V \times V))$.
$(<x, y> \in E) \rightarrow (x \in y)$.
$(<x, y> \in (V \times V)), (x \in y) \rightarrow (<x, y> \in E)$.

Axiom B-2: $\cap$ (binary intersection).
$\forall z\ \forall x\ \forall y((z \in (x \cap y)) \Longleftrightarrow (z \in x) \ \&\ (z \in y))$.

$(z \in (x \cap y)) \rightarrow (z \in x)$.
$(z \in (x \cap y)) \rightarrow (z \in y)$.
$(z \in x), (z \in y) \rightarrow (z \in (x \cap y))$.

Axiom B-3: ~ (complement).
$\forall z\ \forall x((z \in \sim(x)) \Longleftrightarrow (z \in V) \ \&\ \neg(z \in x))$.

$(z \in \sim(x)), (z \in x) \rightarrow$.
$(z \in V) \rightarrow (z \in \sim(x)), (z \in x)$.

Theorem (SP2) in Appendix 2 introduces the null class *0*.

Definition of $\cup$ (binary union).
$\forall x\ \forall y((x \cup y) = \sim((\sim(x) \cap \sim(y))))$.

$\rightarrow (\sim((\sim(x) \cap \sim(y))) = (x \cup y))$.

Definition of + (symmetric difference).
$\forall x\ \forall y((x + y) = (\sim(x \cap y) \cap \sim(\sim(x) \cap \sim(y))))$.

$\rightarrow ((\sim((x \cap y)) \cap \sim((\sim(x) \cap \sim(y)))) = (x + y))$.

Definition of *restrict* (restriction).
$\forall x(restrict(xr, x, y) = (xr \cap (x \times y)))$.

$\rightarrow ((xr \cap (x \times y)) = restrict(xr, x, y))$.

Axiom B-4: *D* (domain).
$\forall x\ \forall z((z \in D(x)) \Longleftrightarrow (z \in V) \ \&\ \neg(restrict(x, \{z\}, V) = 0))$.

$(restrict(x, \{z\}, V) = 0), (z \in D(x)) \rightarrow$.
$(z \in V) \rightarrow (restrict(x, \{z\}, V) = 0), (z \in D(x))$.

Axiom B-7: *rotate*.
$\forall x(rotate(x) \subseteq ((V \times V) \times V))$.
$\forall x\ \forall u \forall v \forall w((<<u, v>, w> \in rotate(x)) \Longleftrightarrow (<<u, v>, w> \in ((V \times V) \times V)) \ \&\ (<<v, w>, u> \in x))$.

$\rightarrow (rotate(x) \subseteq ((V \times V) \times V))$.
$(<<u, v>, w> \in rotate(x)) \rightarrow (<<v, w>, u> \in x)$.
$(<<v, w>, u> \in x), (<<u, v>, w> \in ((V \times V) \times V)) \rightarrow (<<u, v>, w> \in rotate(x))$.

Axiom B-8: *flip*.
$\forall x(flip(x) \subseteq ((V \times V) \times V))$.
$\forall z\ \forall u\ \forall v\ \forall w((<<u, v>, w> \in flip(x)) \Longleftrightarrow (<<u, v>, w> \in ((V \times V) \times V)) \ \&\ (<<v, u>, w> \in x))$.

$\rightarrow$ $(flip(x) \subseteq ((V \times V) \times V))$.
$(<<u, v>, w> \in flip(x))$ $\rightarrow$ $(<<v, u>, w> \in x)$.
$(<<v, u>, w> \in x), (<<u, v>, w> \in ((V \times V) \times V))$ $\rightarrow$ $(<<u, v>, w> \in flip(x))$.

Definition of *inverse*.
$\forall y(inverse(y) = D(flip((y \times V))))$.

$\rightarrow$ $(D(flip((y \times V))) = inverse(y))$.

Definition of *R* (range).
$\forall z(R(z) = D(inverse(z)))$.

$\rightarrow$ $(D(inverse(z)) = R(z))$.

Definition of *dom*.
$\forall z\ \forall x\ \forall y(dom(z, x, y) = 1st(notsub(restrict(z, x, \{y\}), 0)))$.

$\rightarrow$ $(1st(notsub(restrict(z, x, \{y\}), 0)) = dom(z, x, y))$.

Definition of *ran*.
$\forall z\ \forall x(ran(z, x, y) = 2nd(notsub(restrict(z, \{x\}, y), 0)))$.

$\rightarrow$ $(2nd(notsub(restrict(z, \{x\}, y), 0)) = ran(z, x, y))$.

Definition of " (image).
$\forall x\ \forall xr((xr \text{ " } x) = R(restrict(xr, x, V)))$.

$\rightarrow$ $(R(restrict(xr, x, V)) = (xr \text{ " } x))$.

Definition of *succ* (successor).
$\forall x(succ(x) = (x \cup \{x\}))$.

$\rightarrow$ $((x \cup \{x\}) = succ(x))$.

Definition of *SUCC* from the class existence theorem.
$(SUCC \subseteq (V \times V))$.
$\forall x\ \forall y((<x, y> \in SUCC) \iff (<x, y> \in (V \times V)\ \&\ (succ(x) = y)))$.

$\rightarrow$ $(SUCC \subseteq (V \times V))$.
$(<x, y> \in SUCC)$ $\rightarrow$ $(succ(x) = y)$.
$(succ(x) = y), (x \in V)$ $\rightarrow$ $(<x, y> \in SUCC)$.

Definition of *INDUCTIVE*.
$\forall x(INDUCTIVE(x) \iff 0 \in x\ \&\ (SUCC \text{ " } x) \subseteq x))$.

$INDUCTIVE(x)$ $\rightarrow$ $(0 \in x)$.
$INDUCTIVE(x)$ $\rightarrow$ $((SUCC \text{ " } x) \subseteq x)$.
$(0 \in x), ((SUCC \text{ " } x) \subseteq x)$ $\rightarrow$ $INDUCTIVE(x)$.

Axiom C-1: Infinity.
$\exists x((x \in V)\ \&\ INDUCTIVE(x)\ \&\ \forall y(INDUCTIVE(y) \Rightarrow (x \subseteq y)))$.

$\rightarrow$ $INDUCTIVE(\omega)$.
$INDUCTIVE(y)$ $\rightarrow$ $(\omega \subseteq y)$.
$\rightarrow$ $(\omega \in V)$.

Definition of *U* (sum class).
$\forall x(U(x) = D(restrict(E, V, x)))$.

$\rightarrow$ $(D(restrict(E, V, x)) = U(x))$.

Axiom C-2: *U* (sum class).
$\forall x((x \in V) \Rightarrow (U(x) \in V))$.

$(x \in V) \rightarrow (U(x) \in V)$.

Definition of P (power class).
$\forall x(P(x) = \sim((E \text{ “ } \sim(x))))$.

$\rightarrow (\sim((E \text{ “ } \sim(x))) = P(x))$.

Axiom C-3: P (power class).
$\forall u((u \in V) \Rightarrow (P(u) \in V))$.

$(u \in V) \rightarrow (P(u) \in V)$.

Definition of $\circ$ (composition).
$\forall xr\ \forall yr((yr \circ xr) \subseteq (V \times V))$.
$\forall u\ \forall v\ \forall xr\ \forall yr((<u, v> \in (yr \circ xr)) \iff (<u, v> \in (V \times V)) \,\&\, (v \in (yr \text{ “ } (xr \text{ “ } \{u\}))))$.

$\rightarrow ((yr \circ xr) \subseteq (V \times V))$.
$(<y, z> \in (yr \circ xr)) \rightarrow (z \in (yr \text{ “ } (xr \text{ “ } \{y\})))$.
$(z \in (yr \text{ “ } (xr \text{ “ } \{y\}))), (<y, z> \in (V \times V)) \rightarrow (<y, z> \in (yr \circ xr))$.

Definition of *SINGVAL* (single-valued class).
$\forall x(SINGVAL(x) \iff ((x \circ inverse(x)) \subseteq I)$.

$SINGVAL(x) \rightarrow ((x \circ inverse(x)) \subseteq I)$.
$((x \circ inverse(x)) \subseteq I) \rightarrow SINGVAL(x)$.

Definition of *FUNCTION*.
$\forall xf(FUNCTION(xf) \iff (xf \subseteq (V \times V)) \,\&\, SINGVAL(xf))$.

$FUNCTION(xf) \rightarrow (xf \subseteq (V \times V))$.
$FUNCTION(xf) \rightarrow SINGVAL(xf)$.
$(xf \subseteq (V \times V)), SINGVAL(xf) \rightarrow FUNCTION(xf)$.

Axiom C-4: Replacement.
$\forall x((x \in V) \,\&\, FUNCTION(xf) \Rightarrow ((xf \text{ “ } x) \in V))$.

$FUNCTION(xf), (x \in V) \rightarrow ((xf \text{ “ } x) \in V)$.

Axiom D: Regularity.
$\forall x(\neg(x = 0) \Rightarrow \exists u((u \in V) \,\&\, (u \in x) \,\&\, ((u \cap x) = 0)))$.

$\rightarrow (x = 0), (regular(x) \in x)$.
$\rightarrow (x = 0), ((regular(x) \cap x) = 0)$.

Definition of ‘ (functional application).
$\forall xf\ \forall y((xf \text{ ‘ } y) = U((xf \text{ “ } \{y\})))$.

$\rightarrow (U((xf \text{ “ } \{y\})) = (xf \text{ ‘ } y))$.

Axiom E: Universal choice
$\exists xf(FUNCTION(xf) \,\&\, \forall y((y \in V) \Rightarrow (y = 0) \vee ((xf \text{ ‘ } y) \in y)))$.

$\rightarrow FUNCTION(choice)$.
$(y \in V) \rightarrow (y = 0), ((choice \text{ ‘ } y) \in y)$.

GROUP 2: MORE SET THEORY DEFINITIONS.

Definition of *ONEONE* (one-to-one function)
$\forall xf(ONEONE(xf) \iff FUNCTION(xf) \,\&\, FUNCTION(inverse(xf)))$.

$ONEONE(xf) \rightarrow FUNCTION(xf)$.

$ONEONE(xf) \rightarrow FUNCTION(inverse(xf)).$
$FUNCTION(inverse(xf)), FUNCTION(xf) \rightarrow ONEONE(xf).$

Definition of S (subset relation).
$(S = (\sim((\sim(E) \circ inverse(E))) \cap (V \times V)).$

$\rightarrow ((\sim((\sim(E) \circ inverse(E))) \cap (V \times V)) = S).$

Definition of I (identity relation).
$(I = (S \cap inverse(S))).$

$\rightarrow ((S \cap inverse(S)) = I).$

Definition of *diag* (diagonalization).
$\forall xr(diag(xr) = \sim(D((I \cap xr)))).$

$\rightarrow (\sim(D((I \cap xr))) = diag(xr)).$

Definition of Cantor class.
$\forall x(cantor(x) = (D(x) \cap diag((inverse(E) \circ x)))).$

$\rightarrow ((D(x) \cap diag((inverse(E) \circ x))) = cantor(x)).$

Definition of *OPERATION*.
$\forall xf(OPERATION(xf) \Longleftrightarrow FUNCTION(xf) \ \& \ ((D(D(xf)) \times D(D(xf))) = D(xf)) \ \& \ (R(xf) \subseteq D(D(xf))).$

$OPERATION(xf) \rightarrow FUNCTION(xf).$
$OPERATION(xf) \rightarrow ((D(D(xf)) \times D(D(xf))) = D(xf)).$
$OPERATION(xf) \rightarrow (R(xf) \subseteq D(D(xf))).$
$FUNCTION(xf), ((D(D(xf)) \times D(D(xf))) = D(xf)), (R(xf) \subseteq D(D(xf))) \rightarrow OPERATION(xf).$

Definition of *COMPATIBLE*.
$\forall xh \ \forall xf \ \forall xg(COMPATIBLE(xh, xf, xg) \Longleftrightarrow FUNCTION(xh) \ \& \ (D(D(xf)) = D(xh)) \ \& \ (R(xh) \subseteq D(D(xg)))).$

$COMPATIBLE(xh, xf, xg) \rightarrow FUNCTION(xh).$
$COMPATIBLE(xh, xf, xg) \rightarrow (D(D(xf)) = D(xh)).$
$COMPATIBLE(xh, xf, xg) \rightarrow (R(xh) \subseteq D(D(xg))).$
$FUNCTION(xh), (D(D(xf)) = D(xh)), (R(xh) \subseteq D(D(xg))) \rightarrow COMPATIBLE(xh, xf, xg).$

Definition of *HOM* (homomorphism).
$\forall xh \ \forall xf \ \forall xg(HOM(xh, xf, xg) \Longleftrightarrow OPERATION(xf) \ \& \ OPERATION(xg) \ \& \ COMPATIBLE(xh, xf, xg) \ \&$
$\quad \forall x \ \forall y((<x, y> \in D(xf)) \Rightarrow ((xg \text{‘} <(xh \text{‘} x), (xh \text{‘} y)>) = (xh \text{‘} (xf \text{‘} <x, y>)))).$

$HOM(xh, xf, xg) \rightarrow OPERATION(xf).$
$HOM(xh, xf, xg) \rightarrow OPERATION(xg).$
$HOM(xh, xf, xg) \rightarrow COMPATIBLE(xh, xf, xg).$
$HOM(xh, xf, xg), (<x, y> \in D(xf)) \rightarrow ((xg \text{‘} <(xh \text{‘} x), (xh \text{‘} y)>) = (xh \text{‘} (xf \text{‘} <x, y>))).$
$OPERATION(xf), OPERATION(xg), COMPATIBLE(xh, xf, xg)$
$\quad \rightarrow (<nothom1(xh, xf, xg), nothom2(xh, xf, xg)> \in D(xf)), HOM(xh, xf, xg).$
$OPERATION(xf), OPERATION(xg), COMPATIBLE(xh, xf, xg),$
$\quad ((xg \text{‘} <(xh \text{‘} nothom1(xh, xf, xg)), (xh \text{‘} nothom2(xh, xf, xg))>) =$
$\quad (xh \text{‘} (xf \text{‘} <nothom1(xh, xf, xg), nothom2(xh, xf, xg)>))) \rightarrow HOM(xh, xf, xg).$

5. Proving Classes Equal

Appendix 2 includes a list of theorems of Boolean algebra that I proved using OTTER. Most of these proofs involve showing that two terms denote the same class.

The simplification I provided in Section 3.2 still leaves a difficulty in proving equality of classes. I will discuss this difficulty with specific reference to OTTER, which implements the inference rules hyperresolution, UR-resolution, and binary resolution. The difficulty is that to prove an equality $\rightarrow (a = b)$ for (ground) terms a and b, we must resolve

(E1) $(a = b) \rightarrow$ with

(E2) $(x \subseteq y), (y \subseteq x) \rightarrow (x = y)$ to get

(E3) $(a \subseteq b), (b \subseteq a) \rightarrow$.

However, neither hyperresolution nor UR-resolution will make this inference. Here we clearly want some case analysis mechanism, to split the proof into separate proofs that $\rightarrow (a \subseteq b)$ and $\rightarrow (b \subseteq a)$.

This difficulty may be addressed in at least the following ways:

Unacceptable

1. We can make the desired inference using binary resolution. But binary resolution normally generates far too many unwanted conclusions to be used as an inference rule.
2. Lusk and Overbeek [10] solved this problem by using qualifying literals, and locking to force case analysis. But neither qualification nor locking are currently available in OTTER.
3. Winker and Wos [20] addressed this problem by introducing demodulators (rewrite rules) that define propositional calculus at the term level. They would introduce a function EL taking on values T and F, and have clauses such as

   $(EL(x, y) = T) \rightarrow (x \in y)$,

   three demodulators defining $\cup$, $\cap$, and $\sim$ such as

   $\rightarrow (EL(x, (y \cup z)) = OR(EL(x, y), EL(x, z)))$,

   plus about 37 propositional logic demodulators such as

   $\rightarrow (NOT(NOT(x)) = x)$

   $\rightarrow (OR(x, T) = T)$.

   Thus they reduce proving theorems of Boolean algebra to using propositional logic rewrite rules.

   If we were to follow this approach, we would presumably adopt some of the propositional logic demodulators as axioms, and prove the rest of them. But this is more or less the same amount of work as proving the theorems of Boolean algebra directly. Furthermore, these demodulators only seem useful for the duration of proving the corresponding theorems of Boolean algebra such as

   $\rightarrow (\sim(\sim(x)) = x)$,

   $\rightarrow ((x \cup V) = V)$;

   henceforth we would rely on the Boolean algebra rewrite rules.

   All told, it does not seem of net benefit to introduce this temporary machinery.

Marginally Acceptable

4. We prove the following four theorems by resolving (E2) against the definition of the subclass relation:

   $\rightarrow (x = y), (notsub(x, y) \in x), (notsub(y, x) \in y).$

   $(notsub(x, y) \in y) \rightarrow (x = y), (notsub(y, x) \in y).$

   $(notsub(y, x) \in x) \rightarrow (x = y), (notsub(x, y) \in x).$

   $(notsub(x, y) \in y), (notsub(y, x) \in x) \rightarrow (x = y).$

   Now a proof that $\rightarrow (a = b)$ can proceed by hyperresolution. In particular, we first deduce

   (E4) $\rightarrow (notsub(a, b) \in a), (notsub(b, a) \in b).$

   The drawback to this approach is that (E4) contains literals from both of the cases $a \subseteq b$ and $b \subseteq a$; these cases are considered simultaneously, which is not as efficient as considering them serially. It also requires us to use hyperresolution, whereas it is usually more efficient to obtain proofs by UR-resolution alone.
5. The user undertakes the case analysis, by separately proving the two theorems $\rightarrow (a \subseteq b)$ and $\rightarrow (b \subseteq a)$.

   Note that this approach should be needed less and less as the development proceeds. Once we have built up a body of theorems, we may often expect to prove $\rightarrow (a = b)$ from other known equalities, without having to open up the basic definition of equality.

I used Method 4 to prove all the theorems of Boolean algebra, except the distributive law, which I proved using Methods 4 and 5. The proof of Theorem (12) below shows how Method 4 is used to prove $\rightarrow ((x \cap y) = (y \cap x))$.

Theorem (I2) $\rightarrow (x \cap y) = (y \cap x).$

Axioms and Previously Proven Theorems:

```
 23 (z ∈ (x ∩ y)) → (z ∈ x).
 24 (z ∈ (x ∩ y)) → (z ∈ y).
 25 (z ∈ x), (z ∈ y) → (z ∈ (x ∩ y)).
153  → (x = y), (notsub(x, y) ∈ x), (notsub(y, x) ∈ y).
154 (notsub(x, y) ∈ y) → (x = y), (notsub(y, x) ∈ y).
155 (notsub(y, x) ∈ x) → (x = y), (notsub(x, y) ∈ x).
156 (notsub(x, y) ∈ y), (notsub(y, x) ∈ x) → (x = y).
```

Negation of Theorem:

```
157 ((a ∩ b) = (b ∩ a)) → .
```

The Proof:

```
164 → (notsub((a ∩ b), (b ∩ a)) ∈ (a ∩ b)),
      (notsub((b ∩ a), (a ∩ b)) ∈ (b ∩ a))        [hyper, 157, 153].
166 → (notsub((b ∩ a), (a ∩ b)) ∈ (b ∩ a)),
      (notsub((a ∩ b), (b ∩ a)) ∈ b)              [hyper, 164, 24].
167 → (notsub((b ∩ a), (a ∩ b)) ∈ (b ∩ a)),
      (notsub((a ∩ b), (b ∩ a)) ∈ a)              [hyper, 164, 23].
```

```
181  → (notsub((b ∩ a), (a ∩ b)) ∈ (b ∩ a)),
       (notsub((a ∩ b), (b ∩ a)) ∈ (b ∩ a))      [hyper, 167, 25, 166].
240  → (notsub((b ∩ a), (a ∩ b)) ∈ (b ∩ a))      [hyper, 181, 154, unit_del, 157].
242  → (notsub((b ∩ a), (a ∩ b)) ∈ a)            [hyper, 240, 24].
243  → (notsub((b ∩ a), (a ∩ b)) ∈ b)            [hyper, 240, 23].
248  → (notsub((b ∩ a), (a ∩ b)) ∈ (a ∩ b))      [hyper, 243, 25, 242].
251  → (notsub((a ∩ b), (b ∩ a)) ∈ (a ∩ b))      [hyper, 248, 155, unit_del, 157].
261  → (notsub((a ∩ b), (b ∩ a)) ∈ b)            [hyper, 251, 24].
262  → (notsub((a ∩ b), (b ∩ a)) ∈ a)            [hyper, 251, 23].
269  → (notsub((a ∩ b), (b ∩ a)) ∈ (b ∩ a))      [hyper, 262, 25, 261].
273  → ((a ∩ b) = (b ∩ a))                       [hyper, 269, 156, 248].
282  →                                           [binary, 273, 157].
```

6. Boolean Demodulators

If we use the standard operation symbols '$\sim$', '$\cup$', and '$\cap$', it is difficult to know what set of rewrite rules to incorporate. There is no complete set of reductions for free Boolean algebras, since there may be more than one minimal set of prime implicants [18].

Alternatively, we may eliminate '$\cup$' in favor of the symmetric difference functor '$+$' by using

$$x \cup y \;=\; x + y + (x \cap y).$$

We can then use the complete set of reductions derived from those for a Boolean ring, and thus reduce every Boolean term to a unique form. The drawback to this approach is that it is more natural to express theorems in terms of '$\cup$' than in terms of '$+$'. The approach of [7] also requires us to eliminate the complement functor '$\sim$' via

$$\sim x \;=\; x + V.$$

Below I supply such a complete set of reductions. I have modified the reductions given in [7] in the following respects:

1. To keep the canonical form as nearly natural as possible, I have retained the '$\sim$' functor. The rules will reduce any term containing '$\sim$' to one in which '$\sim$' appears at most once, and that appearance is initial. Correspondingly, 'V' will never appear as a summand in a reduced term.
2. The reductions in [7] are complete with respect to associative-commutative unification. Since neither of these unification options are available in OTTER, we must use lex-dependent demodulators to write each term as a unique right-associated sum of products. We also need to add a demodulator under (I6) and one under (E6), to effect the reductions from this lex-ordered right-associated form.

BOOLEAN DEMODULATORS REPLACING UNION BY SYMMETRIC DIFFERENCE

These demodulators should be used with the lexical order specifier

$$lex([0,\ V,\ \sim(x),\ (x \cap x),\ (x \cup x),\ (x + x)]).$$

To assure uniqueness of the canonical form, we also should select the OTTER option *lex_order_vars*, which defines and uses a lexical ordering of variables.

INTERSECTION

(I2) Commutative law of intersection.
$\rightarrow ((x \cap y) = (y \cap x))$.

(I3) Lexical ordering within associations.
$\rightarrow ((x \cap (y \cap z)) = (y \cap (x \cap z)))$.

(I4) Intersection with 0.
$\rightarrow ((0 \cap x) = 0)$.

(I5) Intersection with V.
$\rightarrow ((V \cap x) = x)$.

(I6) Idempotent law of intersection.
$\rightarrow ((x \cap x) = x)$.
Corollary.
$\rightarrow ((x \cap (x \cap y)) = (x \cap y))$.

COMPLEMENT

(C1) Complement of complement.
$\rightarrow (\sim(\sim(x)) = x)$.

(C2) Special cases.
$\rightarrow (\sim(0) = V)$.
$\rightarrow (\sim(V) = 0)$.

(C3) Elimination of complement in intersection.
$\rightarrow ((\sim(x) \cap y) = (y + (x \cap y)))$.

(C4) Moving complement out in symmetric difference.
$\rightarrow ((\sim(x) + y) = \sim((x + y)))$.

SYMMETRIC DIFFERENCE

(E1) Elimination of union.
$\rightarrow ((x \cup y) = ((x \cap y) + (x + y)))$.

(E2) Commutative law of symmetric difference.
$\rightarrow ((x + y) = (y + x))$.

(E3) Lexical ordering within associations.
$\rightarrow ((x + (y + z)) = (y + (x + z)))$.

(E4) Symmetric difference with 0.
$\rightarrow ((0 + x) = x)$.

(E5) Symmetric difference with V.
$\rightarrow ((V + x) = \sim(x))$.

(E6) Nilpotent law of symmetric difference.
$\rightarrow ((x + x) = 0)$.
Corollary.
$\rightarrow ((x + (x + y)) = y)$.

DISTRIBUTIVE LAW

(D1) Intersection distributes over symmetric difference.
$\rightarrow ((x \cap (y + z)) = ((x \cap y) + (x \cap z)))$.

These demodulators should be useful in any study that heavily involves Boolean algebra. But for the further development reported in this paper, I continue to use '$\cup$' rather than '$+$'. I found the Boolean demodulators used rarely, and that those given in Appendix 2 more than sufficed. With that incomplete set of reductions, it is not clear which orientation of the distributive laws is most useful. In Appendix 2 I have oriented them to factor rather than distribute.

7. Skolem Functors and the Axiom of Choice

Skolem functors serve much the same purpose as Hilbert ε-terms – they provide names for objects that are guaranteed to exist by existential assertions. Unless there is exactly one object satisfying the existential assertion, the object named is indeterminate.

Since equality is ubiquitous, the Skolem functor *notsub* appearing in the definition of the subclass relation sees much duty in our system. Note that *notsub*$(x, 0)$ acts as a *universal choice term*, in that we have the Theorem (SP4):

$$\rightarrow (x = 0), (\mathit{notsub}(x, 0) \in x). \tag{1}$$

Several textbooks in set theory (for example, [2]) give, as a very strong version of the Axiom of Choice, the Axiom of Global Choice:

$$c \neq 0 \Rightarrow \sigma(c) \in c, \tag{2}$$

where 'c' is a variable restricted to sets, and 'σ' is a unary functor. Since we already have such a functor in (1), why do we need Axiom E of Choice in addition?

The answer is that set theories using (2) also have an Axiom Schema of Replacement which allows σ to appear in the formula of the schema. In Gödel's system the corresponding schema is the class existence theorem, which only applies to formulas of the basic theory in which there are no functors. In particular it does not apply to formulas containing the Skolem functor '*notsub*', so there is no way to prove in our clausal system that the intuitively conceived class $\{\langle x, \mathit{notsub}(x, 0)\rangle : x \in V\}$ exists. That is, there is no way to reify the term *notsub*$(x, 0)$ into a function, whereas Axiom E explicitly assumes the existence of such a function.

I have introduced several Skolem functors, such as '*1st*' and '*2nd*', via uniqueness theorms of the form $\vdash(\forall x_1 \ldots \forall x_n)\,(\exists! y)\phi$. The uniqueness theorems guarantee that such functors are eliminable, and thus they may be included in formulas to which we apply the class existence theorem. But there is no way to define *notsub*$(x, 0)$ uniquely, or else we could prove the Axiom of Choice!

We see that the real power of the Axiom of Choice is not that it allows one to make infinitely many arbitrary selections, but that it asserts the existence either of a set (or class) containing the selected entities, of the corresponding function (class of ordered pairs) that makes the choices.

8. Theorems Proved

Appendix 2 contains a list of most of the theorems I have proved to date from the above axioms. I selected these theorems while producing my own development of elementary set theory; they turn out to constitute a fair sampling of the theorems found in the first 90 pages of Suppes [19].

These theorems were proved automatically, using OTTER. The proofs were not hand-guided as in [4]. Since OTTER is not interactive, user guidance comes principally from assignment of weights and selection of inference rules *before* a proof run, as described in Section 10. It is also up to the user to organize the sequence of theorems proved so that a proof attempt can build upon previous results without biting off too large a new chunk.

9. Use of Previously Proved Theorems

Bledsoe [3] obtained proofs of theorems in set theory by only loading those reference theorems needed in the proof. He pointed out that "A different set of reference theorems would have yielded a different proof or no proof, and too large a set would have lead to no proof at all".

If I only loaded reference theorems needed in a proof, I believe that almost all the 400-plus theorems in Appendix 2 would have been proved very easily (see, for example, the end of Section 13). Adopting such a strategy trivializes the problem, at least using OTTER on the theorems listed. Since I am conducting a systematic development of set theory, when I prove a theorem I add its clausal version to the Axiom list (see Appendix 2). I always load *all* previously proved theorems. The computer should be able to bring to bear everything it knows in proving the theorem at hand. An exception is that I do not load the Axiom of Choice. It is not needed in elementary set theory, and it is bad form to use it when it is not needed. Of course I also do not load the few theorems that are subsumed by later theorems (see the beginning of Appendix 2).

Thus, along with a few theorems that are relevant to the proof of the theorem at hand, I load many irrelevant theorems that act as noise in producing many irrelevant conclusions. To overcome that problem, I frequently find it necessary to intervene to a much lesser extent than only loading relevant theorems. Instead I assign low weights to functors I expect must appear in the proof (see Section 10). Proof runs then still produce many irrelevant conclusions, but fewer of them move to the top of the set of support to be further considered.

10. Heuristics and Option Settings

The weighting option in OTTER is described in Section A11 of Appendix 1. I assign weight 1 to every term, including every subterm, appearing in the theorem. Without this heuristic almost all the proof attempts would have run on until user termination. This weighting heuristic is so universally useful that I recommend its incorporation into OTTER as an auto-weighting option. It is a first step toward an *automated* determination of optimal weight assignment. The next step to explore is the automatic assignment of low weights to all 'sufficiently similar' terms; for example if $f(a)$ is assigned weight 1, should all terms $f(x)$ also be assigned weight 1?

If the conclusion of the theorem is of the form $(t1 \subseteq t2)$, I also use the template

$weight(notsub(1, 1), -1)$.

Most of the theorems I have proved are universal statements, in which the clausal version of the negation of the theorem contains no free variables. In such cases I also assign a high enough weight to variables so that all deduced clauses containing variables will be discarded. Thus any proof obtained will be a ground proof. This heuristic is successful in the large majority of cases.

The set-of-support strategy, the assignment of low weights to terms in theorems, and the discarding of formulas containing free variables, all work to keep OTTER focusing upon the objects named in the theorem. In this respect the object-oriented inference mechanisms used by McAllester in the system Ontic [8] are strongly analogous to my rules.

For use with lex-dependent demodulators, I use the lexical specifier

lex([*0*, *V*, $\sim(x)$,
a, *b*, *c*, *d*, *f*, *g*, *g1*, *g2*, *h*, *h1*, *h2*,
notsub(x, x), $\{x\}$, *memb*(x), *memb'*(x), $\{x, x\}$,
$\langle x, x \rangle$, $(x \times x)$, *1st*(x), *2nd*(x), *1st'*(x),
2nd'(x), *E*, *restrict*(x, x, x), *D*(x), *rotate*(x), *flip*(x), *inverse*(x), *R*(x),
dom(x, x, x), *ran*(x, x, x), (x " x) *succ*(x), *SUCC*, ω, *U*(x), *P*(x), $(x \circ x)$,
sv1(x), *sv2*(x), *sv3*(x), *SS*, *regular*(x), (x 'x), *choice*,
S, *I*, *diag*(x), *cantor*(x), *nothom1*(x, x, x), *nothom2*(x, x, x),
$(x \cap x)$, $(x \cup x)$, $(x + x)$]),

where the order is from most preferred to least preferred. The first and last lines of this specification are significant. The second line consists of Skolem constants used in expressing the negation of a theorem. The order of the remaining functors matters little.

In my first attempt at a proof, I turn on UR-resolution, paramodulation into and from, and back demodulation. If these settings fail to obtain a proof within a reasonable length of time, I try turning on hyperresolution. I then also assign low weights to other functors that I expect must appear in the proof. This step is frequently necessary.

Suppose, as is frequently the case, that the theorem we are trying to prove is a Horn clause. If one puts *all* literals from the theorem in the set of support, then the proof will consist of some true statements (in any model of the hypotheses) that follow from the hypotheses, plus some false statements that follow from the negation of the conclusion. The user may find it confusing to look through such a proof and keep in mind which statements he is to believe, and which he is to disbelieve.

One doesn't have this problem while reading a proof in which every deduced line is true. Thus I prefer proofs that proceed in a completely forward direction from the hypotheses of the theorem, and only use the negation of the conclusion in the last line where the contradiction is recognized (and which step is then superfluous). One might expect such proofs to be difficult to obtain, because the prover is not making any use of the goal for guidance. Nonetheless many, perhaps the majority, of the proofs of the theorems listed in this paper were obtained in this way. I do this by putting the negation of the conclusion in the axiom list, outside of the set of support.

An especially pleasing subset of such proofs is formed by those in which (almost) every deduction line consists of a positive literal obtained by UR-resolution, in that negative deduced literals are one level more psychologically complex than positive literals. To assist in obtaining such proofs, one can assign a high weight to the negation sign, so that the prover will focus on positive (unit) clauses. I did that in obtaining all but one of the proofs presented in Sections 12 and 13. Without that weight template, the contradiction in the proof of Theorem (CO9) below comes at line 8179 and takes 375.53 seconds.

I have not tested this last heuristic extensively enough to know how widely applicable it is. It is unlikely to work if the negated theorem contains a positive literal of the form $\rightarrow(x \in \sim(y))$, since this probably will have to be converted to $(x \in y) \rightarrow$. It also will not work when the conclusion contains Skolem constants not appearing in the hypothesis, since the forward proof would require derived clauses containing variables, which I discard. Otherwise, it certainly appears promising.

I never use binary resolution, since it generates far too much junk to use regularly in automated proofs. Rather than succumb to the temptation to use binary resolution when other methods fail, one should instead try to figure out what changes need to be made so that more efficient resolution rules may be used.

The heuristics presented above have evolved from those presented in [16] and [17].

11. Proof Finder or Proof Verifier?

My heuristic of assigning weight 1 to terms appearing in the theorem statement normally suffices to obtain proofs in which no other functors appear. But if other functors are needed in the proof, it is often necessary to assign them low weights as well.

Theorems in this paper typically required a handful of computer runs to obtain a proof. The most difficult theorem, the composition of homomorphisms theorem in Section 13, took more than 20 runs spread over a week. When it was necessary to add weight templates, I could do so without great difficulty, since I normally had an idea of how the proof should proceed – or would work out a proof sketch while watching OTTER fail. Sometimes it was necessary to backtrack and ask OTTER to prove a simpler theorem first. Further, if OTTER obtained a proof that was longer or less elegant than my hand proof, I would tinker with the settings for several more runs until it obtained my proof.

The continuum between proof finder and proof verifier is largely measured by the degree of user guidance supplied to the prover. Almost all my heuristics and parameter settings are or can be made automatic, except for the frequent need to supply additional weighting templates. When such additional templates are necessary, I call the proof *semiautomatic*.

Of course, if we are asking OTTER to act solely as a proof *verifier*, we can automatically supply low weights to all terms appearing in the proof outline, and expect that OTTER will then normally find the full proof. Remember also that even when the user supplies weights in advance, OTTER must still proceed without further guidance to generate proofs requiring up to about 30 deduction steps.

Thus we may say that the main obstacle to OTTER acting as a fully automatic proof finder in set theory is automation of the method of selecting weights. Wos [22] presents this as Problem 27. It is closely related to the problem of automatically determining which of a large body of previously proved theorems are most relevant to the proof of the theorem at hand.

12. Proof of Cantor's Theorem

The next two sections contain several OTTER proofs. The proof traces show only the axioms and the deduction steps that actually contribute to the proof. For example, in the proof immediately below, 352 previously proved theorems were loaded but only 8 used in the proof. There were at least 269 retained conclusions, but only 8 of these contributed to the proof. While I do not turn on binary resolution as an option, OTTER uses binary resolution to obtain the final unit conflict.

I now prove Cantor's theorem that there can be no function mapping a set onto its power set. I first define the Cantor class by

$$cantor(x) \;=\; D(x) \cap diag((inverse(E) \circ x)),$$

which can be simplifed to

$$cantor(x) \;=\; D(x) \cap \sim D((E \cap x)).$$

I have previously proved the membership conditions 350–352 below of the Cantor class.

Theorem (CA4) $SINGVAL(x), (D(x) \in V), (P(D(x)) \subseteq R(x)) \rightarrow .$

Axioms and Previously Proven Theorems:

2 $(u \in x), (x \subseteq y) \rightarrow (u \in y).$
266 $(y \in R(z)) \rightarrow (dom(z, V, y) \in D(z)).$
289 $(z \in V), (z \subseteq x) \rightarrow (z \in P(x)).$
301 $(y \in V), (x \subseteq y) \rightarrow (x \in V).$
333 $(y \in R(z)), SINGVAL(z) \rightarrow ((z \text{`} dom(z, V, y)) = y).$
350 $\rightarrow (cantor(x) \subseteq D(x)).$
351 $(z \in cantor(xr)), (z \in (xr \text{`} z)) \rightarrow .$
352 $(z \in D(xr)) \rightarrow (z \in (xr \text{`} z)), (z \in cantor(xr)).$

Negation of Theorem:

353 $\rightarrow SINGVAL(f).$

354 → $(P(D(f)) \subseteq R(f))$.
355 → $(D(f) \in V)$.

The Proof:

```
432 → (cantor(f) ∈ V)                          [ur, 355, 301, 350].
458 → (cantor(f) ∈ P(D(f)))                    [ur, 432, 289, 350].
554 → (cantor(f) ∈ R(f))                       [ur, 458, 2, 354].
586 → ((f ' dom(f, V, cantor(f))) = cantor(f)) [ur, 554, 333, 353].
590 → (dom(f, V, cantor(f)) ∈ D(f))            [ur, 554, 266].
696 → (dom(f, V, cantor(f)) ∈ cantor(f))       [para_from, 586, 352, unit_del, 590].
697 (dom(f, V, cantor(f)) ∈ cantor(f)) →       [para_from, 586, 351].
700 →                                          [binary, 696, 697].
```

This proof required 10.95 seconds on a VAX 8800.

Could OTTER *reinvent* the diagonal argument? It could certainly construct the Cantor class 'on the fly', not requiring my previous definition. But it may be too much to expect OTTER to prove its membership conditions 350–352 while trying to prove this theorem. Nonetheless, given a class with these membership conditions, OTTER is able to polish off the proof in a sprightly and natural manner.

13. Proof that the Composition of Homomorphisms is a Homomorphism

Boyer *et al.* offer this theorem as a challenge problem. They then provide a sequence of 27 lemmas that lead to its proof, together with hand-guided proofs of the lemmas.

It is clear that one could not obtain most of their proofs, as presented, by a standard resolution theorem prover using hyperresolution and/or UR-resolution. Here is one line in their proof of Lemma 18, using my notation:

$$(f17b \in V), (\langle f2b, x\rangle \in (V \times V)), (1st(\langle f2b, x\rangle) = f17b),$$
$$(\langle f2b, x\rangle \in V), (f2b \in V), (x \in V), (f3b \in V), (\langle f3b, x\rangle \in V) \rightarrow,$$

where $f2b$, $f3b$, and $f17b$ are abbreviations for other complex terms.

Even using binary resolution, this complex clause would never automatically move to the top of the set of support to be considered again. Nonetheless, I have been able to obtain semiautomatic proofs of the lemmas and of the theorem.

The most difficult of the lemma proofs they present is, in my notation:

Lemma 18: $(R(x) \subseteq D(y)) \rightarrow (D(x) = D(y \circ x)))$.

Their hand-guided proof requires 25 reference theorems, 2 clauses for the denial of the theorem, 65 lines of deduction, and 12 lines of abbreviations, for a total of 94 lines.

I will prove this theorem in two parts, showing inclusion in both directions. We do not need the hypothesis of the theorem for the easier direction. Because of the deep nesting of terms that occurs, I will also introduce abbreviations into OTTER's proof. I produced these abbreviations by hand after OTTER obtained the unabbreviated proof.

Theorem (CO6) → $(D((y \circ x)) \subseteq D(x))$.

Abbreviations:

notsub = notsub(D((g ∘ f)), D(f))

$ran = ran((g \circ f), notsub, V)$
$dom = dom(g, (f \text{“} \{notsub\}), ran)$

Axioms and Previously Proven Theorems:

3 $\rightarrow (notsub(x, y) \in x), (x \subseteq y).$
4 $(notsub(x, y) \in y) \rightarrow (x \subseteq y).$
261 $(x \in D(z)) \rightarrow (\langle x, ran(z, x, V)\rangle \in z).$
262 $(\langle x, y\rangle \in (V \times V)), (\langle x, y\rangle \in z) \rightarrow (x \in D(z)).$
342 $(\langle u, v\rangle \in (xf \circ yf)) \rightarrow (\langle u, dom(xf, (yf \text{“} \{u\}), v)\rangle \in yf).$
343 $(\langle u, v\rangle \in (xf \circ yf)) \rightarrow (\langle u, dom(xf, (yf \text{“} \{u\}), v)\rangle \in (V \times V)).$

Negation of Theorem:

350 $(D((g \circ f)) \subseteq D(f)) \rightarrow .$

The Proof:

434 $(notsub \in D(f)) \rightarrow$	[ur, 350, 4].
435 $\rightarrow (notsub \in D((g \circ f)))$	[ur, 350, 3].
440 $\rightarrow (\langle notsub, ran\rangle \in (g \circ f))$	[ur, 435, 261].
454 $\rightarrow (\langle notsub, dom\rangle \in (V \times V))$	[ur, 440, 343].
455 $\rightarrow (\langle notsub, dom\rangle \in f)$	[ur, 440, 342].
577 $\rightarrow (notsub \in D(f))$	[ur, 455, 262, 454].
590 $\rightarrow$	[binary, 577, 434].

This proof required 10.89 seconds on a VAX 8800.

Theorem (CO9) $(R(x) \subseteq D(y)) \rightarrow (D(x) \subseteq D((y \circ x))).$

Abbreviations:

$notsub = notsub(D(f), D((g \circ f))).$
$ran = ran(f, notsub, V).$
$ranran = ran(g, ran, V).$

Axioms and Previously Proven Theorems:

2 $(u \in x), (x \subseteq y) \rightarrow (u \in y).$
3 $\rightarrow (notsub(x, y) \in x), (x \subseteq y).$
4 $(notsub(x, y) \in y) \rightarrow (x \subseteq y).$
92 $(z \in (yr \circ xr)) \rightarrow (z \in (V \times V)).$
259 $(x \in D(z)) \rightarrow (\langle x, ran(z, x, V)\rangle \in (V \times V)).$
260 $(x \in D(z)) \rightarrow (\langle x, ran(z, x, V)\rangle \in z).$
261 $(\langle x, y\rangle \in (V \times V)), (\langle x, y\rangle \in z) \rightarrow (x \in D(z)).$
267 $(x \in D(z)) \rightarrow (ran(z, x, V) \in R(z)).$
340 $(\langle x, y\rangle \in (V \times V)), (\langle y, z\rangle \in (V \times V)), (\langle x, y\rangle \in xr), (\langle y, z\rangle \in yr) \rightarrow (\langle x, z\rangle \in (yr \circ xr)).$

Negation of Theorem:

347 $\rightarrow (R(f) \subseteq D(g)).$
348 $(D(f) \subseteq D((g \circ f))) \rightarrow .$

The Proof:

432 $(notsub \in D((g \circ f))) \rightarrow$	[ur, 348, 4].
433 $\rightarrow (notsub \in D(f))$	[ur, 348, 3].
437 $\rightarrow (ran \in R(f))$	[ur, 433, 267].
438 $\rightarrow (\langle notsub, ran\rangle \in f)$	[ur, 433, 260].
439 $\rightarrow (\langle notsub, ran\rangle \in (V \times V))$	[ur, 433, 259].
469 $\rightarrow (ran \in D(g))$	[ur, 437, 2, 347].

577	$\rightarrow$ $(\langle ran, ranran\rangle \in g)$	[ur, 469, 260].
578	$\rightarrow$ $(\langle ran, ranran\rangle \in (V \times V))$	[ur, 469, 259].
671	$\rightarrow$ $(\langle notsub, ranran\rangle \in (g \circ f))$	[ur, 578, 340, 438, 577, 439].
791	$\rightarrow$ $(\langle notsub, ranran\rangle \in (V \times V))$	[ur, 671, 92].
873	$\rightarrow$ $(notsub \in D((g \circ f)))$	[ur, 791, 261, 671].
880	$\rightarrow$	[binary, 873, 432].

This proof required 20.56 seconds on a VAX 8800.

Even if we double-count the clauses that are common to the two proofs, the two theorems require 6 lines of abbreviation, 15 reference theorems, 3 clauses for the negations of the theorems, and 19 lines of deduction, for a total of 43 lines. This is less than half the length of the hand proof of Boyer *et al.*, the number of deduction lines is less than one third, and each deduced clause is a unit. While some of the saving is due to splitting the theorem in half, most of it is due to better clauses. Furthermore, OTTER's two proofs are precisely the proofs that I produced by hand, so I consider them very natural.

In addition to my usual weighting heuristics described above, to obtain these proofs I used the templates:

$$weight(dom(1, 1, 1), -2).$$

$$weight(ran(1, 1, 1), -2).$$

$$weight(\langle 1, 1\rangle, -1).$$

$$weight((1 \text{ ``}1), -1).$$

$$weight(\{1\}, 0).$$

As remarked above, use of these templates makes the proofs less than fully automatic. However, a user would easily intuit that these functors will be needed in the proof, and thus assign them low weights.

Lemma 19 of Boyer *et al.*, slightly restated, is that the composition of single-valued classes is single-valued. Their definition of single-valued is the natural one (see (SV1)-(SV3) in Appendix 2), but it generates three Skolem functors. And their hand-guided proof requires 77 lines.

I avoided the Skolem functors by defining

$$SINGVAL(x) \Leftrightarrow (x \circ inverse(x) \subseteq I),$$

and this definition permits an elegant proof of their Lemma 19 using the relational calculus.

Theorem (SV6) $SINGVAL(x), SINGVAL(y) \rightarrow SINGVAL((y \circ x)).$

Axioms and Previously Proven Theorems:

64 $SINGVAL(x) \rightarrow ((x \circ inverse(x)) \subseteq I).$
65 $((x \circ inverse(x)) \subseteq I) \rightarrow SINGVAL(x).$
97 $(x \subseteq y), (y \subseteq z) \rightarrow (x \subseteq z).$
305 $\rightarrow (((xr \circ yr) \circ zr) = (xr \circ (yr \circ zr))).$
310 $(xr \subseteq I) \rightarrow ((zr \circ (xr \circ ur)) \subseteq (zr \circ ur)).$
311 $\rightarrow (inverse((xr \circ yr)) = (inverse(yr) \circ inverse(xr))).$

Negation of Theorem:

375 *SINGVAL((g ∘ f))* → .
376 → *SINGVAL(f).*
377 → *SINGVAL(g).*

The Proof:

456	*(((g ∘ f) ∘ inverse((g ∘ f))) ⊆ I)* →	[ur, 375, 65].
457	→ *((f ∘ inverse(f)) ⊆ I)*	[ur, 376, 64].
458	→ *((g ∘ inverse(g)) ⊆ I)*	[ur, 377, 64].
473	*(((g ∘ f) ∘ (inverse(f) ∘ inverse(g))) ⊆ I)* →	[para_into, 311, 456].
538	*(((g ∘ f) ∘ (inverse(f) ∘ inverse(g))) ⊆ (g ∘ inverse(g)))* →	[ur, 473, 97, 458].
660	*((g ∘ (f ∘ (inverse(f) ∘ inverse(g)))) ⊆ (g ∘ inverse(g)))* →	[para_into, 305, 538].
839	*((g ∘ ((f ∘ inverse(f)) ∘ inverse(g))) ⊆ (g ∘ inverse(g)))* →	[para_into, 305, 660].
1051	*((f ∘ inverse(f)) ⊆ I)* →	[ur, 839, 310].
1060	→	[binary, 1051, 457].

This proof required 19.09 seconds on a VAX 8800.

Theorem (HO1) *HOM(xh1, xf1, xg1), HOM(xh2, xg1, xg2)* → *HOM((xh2 ∘ xh1), xf1, xg2).*

Abbreviations.

n1 = nothom1((h2 ∘ h1), f1, g2)
n2 = nothom2((h2 ∘ h1), f1, g2)

Axioms and Previously Proven Theorems:

67 *FUNCTION(xf)* → *SINGVAL(xf).*
86 *COMPATIBLE(xh, xf, xg)* → *FUNCTION(xh).*
90 *HOM(xh, xf, xg)* → *OPERATION(xf).*
91 *HOM(xh, xf, xg)* → *OPERATION(xg).*
92 *HOM(xh, xf, xg)* → *COMPATIBLE(xh, xf, xg).*
93 *(<x, y> ∈ D(xf)), HOM(xh, xf, xg)*
→ *((xg ‘ <(xh ‘ x), (xh ‘ y)>) = (xh ‘ (xf ‘ <x, y>))).*
94 *COMPATIBLE(xh, xf, xg), OPERATION(xf), OPERATION(xg)*
→ *(<nothom1(xh, xf, xg), nothom2(xh, xf, xg)> ∈ D(xf)), HOM(xh, xf, xg).*
95 *((xg ‘ <(xh ‘ nothom1(xh, xf, xg)), (xh ‘ nothom2(xh, xf, xg))>) =*
(xh ‘ (xf ‘ <nothom1(xh, xf, xg), nothom2(xh, xf, xg)>))),
COMPATIBLE(xh, xf, xg), OPERATION(xf), OPERATION(xg) → *HOM(xh, xf, xg).*
228 *(<u, v> ∈ (x × y))* → *(u ∈ x).*
229 *(<u, v> ∈ (x × y))* → *(v ∈ y).*
373 *(x ∈ D((yf ∘ xf))), SINGVAL(xf)* → *(((yf ∘ xf) ‘ x) = (yf ‘ (xf ‘ x))).*
393 *COMPATIBLE(xh, xf, xg), OPERATION(xf)* → *((D(xh) × D(xh)) = D(xf)).*
396 *(<x, y> ∈ (D(xh) × D(xh))), COMPATIBLE(xh, xf, xg), OPERATION(xf)*
→ *((xf ‘ <x, y>) ∈ D(xh)).*
397 *(<x, y> ∈ (D(xh) × D(xh))), COMPATIBLE(xh, xf, xg)* → *(<(xh ‘ x), (xh ‘ y)> ∈ D(xg)).*
398 *COMPATIBLE(xh1, xf1, xg1), COMPATIBLE(xh2, xg1, xg2)*
→ *COMPATIBLE((xh2 ∘ xh1), xf1, xg2).*

Negation of Theorem:

399 *HOM((h2 ∘ h1), f1, g2)* → .
400 → *HOM(h1, f1, g1).*
401 → *HOM(h2, g1, g2).*

The Proof.

477	→ *COMPATIBLE(h1, f1, g1)*	[ur, 400, 92].
479	→ *OPERATION(f1)*	[ur, 400, 90].

```
495 → COMPATIBLE(h2, g1, g2)                          [ur, 401, 92].
496 → OPERATION(g2)                                   [ur, 401, 91].
512 → ((D(h1) × D(h1)) = D(f1))                       [ur, 477, 393, 479].
513 → FUNCTION(h1)                                    [ur, 477, 86].
525 → SINGVAL(h1)                                     [ur, 513, 67].
533 → COMPATIBLE((h2 ∘ h1), f1, g2)                   [ur, 495, 398, 477].
589 → ((D((h2 ∘ h1)) × D((h2 ∘ h1))) = D(f1))         [ur, 533, 393, 479].
590 ((g2 ' <((h2 ∘ h1) ' n1), ((h2 ∘ h1) ' n2)>) = ((h2 ∘ h1) ' (f1 ' <n1, n2>))) →
                                                      [ur, 533, 95, 479, 496, 399].
591 → (<n1, n2> ∈ D(f1))                              [ur, 533, 94, 479, 496, 399].
597 → ((g1 ' <(h1 ' n1), (h1 ' n2)>) = (h1 ' (f1 ' <n1, n2>)))   [ur, 591, 93, 400].
600 → (<n1, n2> ∈ (D((h2 ∘ h1)) × D((h2 ∘ h1))))     [para_into, 589, 591].
601 → (<n1, n2> ∈ (D(h1) × D(h1)))                   [para_into, 512, 591].
610 → ((f1 ' <n1, n2>) ∈ D((h2 ∘ h1)))                [ur, 600, 396, 533, 479].
611 → (n2 ∈ D((h2 ∘ h1)))                            [ur, 600, 229].
612 → (n1 ∈ D((h2 ∘ h1)))                            [ur, 600, 228].
616 → (<(h1 ' n1), (h1 ' n2)> ∈ D(g1))                [ur, 601, 397, 477].
639 → (((h2 ∘ h1) ' (f1 ' <n1, n2>)) = (h2 ' (h1 ' (f1 ' <n1, n2>))))
                                                      [ur, 610, 373, 525].
646 → (((h2 ∘ h1) ' n2) = (h2 ' (h1 ' n2)))           [ur, 611, 373, 525].
652 → (((h2 ∘ h1) ' n1) = (h2 ' (h1 ' n1)))           [ur, 612, 373, 525].
660 → ((g2 ' <(h2 ' (h1 ' n1)), (h2 ' (h1 ' n2))>) =
      (h2 ' (g1 ' <(h1 ' n1), (h1 ' n2)>)))
                                                      [ur, 616, 93, 401].
701 → ((h2 ' (g1 ' <(h1 ' n1), (h1 ' n2)>)) = ((h2 ∘ h1) ' (f1 ' <n1, n2>)))
                                                      [para_into, 597, 639].
738 → ((g2 ' <((h2 ∘ h1) ' n1), (h2 ' (h1 ' n2))>) =
      (h2 ' (g1 ' <(h1 ' n1), (h1 ' n2)>)))
                                                      [para_into, 652, 660].
774 → ((g2 ' <((h2 ∘ h1) ' n1), ((h2 ∘ h1) ' n2)>) =
      (h2 ' (g1 ' <(h1 ' n1), (h1 ' n2)>)))
                                                      [para_into, 646, 738].
805 → ((g2 ' <((h2 ∘ h1) ' n1), ((h2 ∘ h1) ' n2)>) = ((h2 ∘ h1) ' (f1 ' <n1, n2>)))
                                                      [para_into, 701, 774].
806 →                                                 [binary, 805, 590].
```

This proof required 157.89 seconds on a VAX 8800.

In contrast to all other proofs I have obtained of theorems of set theory, this one is rather unsatisfactory in that I had to use *very* many weight templates to put OTTER on the right track. Without these weights, it appears that OTTER version 1.01 would not have obtained this proof within acceptable limits of computer time. Because of the substantial hints given to OTTER, this particular proof comes closer to a verification than a proof discovery. But even with these initial hints, OTTER had to make it through the deduction steps shown without further interaction.

Normally in a proof requiring equality substitutions, I want to use the back demodulation option. However, I had to turn this option off to obtain this proof. There are various equations relating the domains, such as line 512. If we use this to canonicalize the domains, then line 601 will no longer unify with line 397.

For comparison, I reran Theorem (HO1) with only the 18 'Axioms and Previously Proven Theorems' and 'Negation of Theorem' clauses loaded. I eliminated *all* weights, except the high weight on variables. OTTER found the proof in 4.62 seconds! This is pleasantly surprising, considering the deep nesting of terms that occurs in the derived clauses. It illustrates my claim that the frequent need to set weights is due to my practice of loading all previously proved theorems in each proof run. If the user applies his wisdom to load only the reference theorems needed for the proof, the problem of obtaining automated proofs is trivialized by at least an order of magnitude.

14. Developing a Unification Algorithm Appropriate to NBG Set Theory

Using Robinson's standard unification algorithm, the term $\sim(x)$ will not unify with, for example, an individual constant c. However, I have recently built the law $\sim(\sim(x)) = x$ into OTTER's (Robinson's) unification algorithm, which permits these terms to unify with the substitution $x = \sim(c)$. In very limited testing on 'natural' theorems, this improved algorithm has not produced shorter proofs.

Commutative unification, to treat '$\cup$', '$\cap$', and '$=$' among others, would be a welcome addition to OTTER. Are there other laws of NBG set theory that can profitably be incorporated into the unification algorithm? Would use of a more general unification algorithm permit a resolution theorem prover to reinvent Cantor's diagonal argument?

Bailin [1] has modifed Huet's type theory unification algorithm to provide a semidecision algorithm for the unifiability of two formulas of ZF set theory modulo a sequence of contractions $(x \in \{y: \phi(y)\}) \Leftrightarrow \phi(x)$. His work does not directly apply to NBG set theory, since Axioms B1–B8 have eliminated the need for the set builder and the doubly-recursive definition of terms and formulas. Is there any part of his algorithm that has a useful analog in NBG set theory?

15. Conclusion

No one shall be able to drive us from the paradise that Cantor created for us.
David Hilbert

Boyer *et al.* paint a somewhat pessimistic picture of the possibility of developing set theory (at least using their clauses) with current theorem provers. My experience has been much more positive. With my revised clausal form of the axiom system, along with the heuristics I use, theorems in elementary set theory can be proved semi-automatically without great difficulty. There is no apparent obstacle to the development of set theory through considerably more difficult theorems. In particular the other challenge problems presented in their paper should be within near reach (except the two unsolved problems, which will take a little longer).

Of course, there is a substantial gap in difficulty between reproving known theorems and attacking open problems on the frontiers of research. But we are witnessing a steady increase in the intelligence of theorem proving software. More dramatically, the number of computations per dollar obtainable by computer hardware is approximately doubling every two years. Moravec [14] extrapolates that \$1000 personal computers with the computational power of the human brain should be available by year 2030. The time will come when such crushers as Riemann's hypothesis and

Goldbach's conjecture will be fair game for automated reasoning programs. For those of us who arrange to stick around, endless fun awaits us in the automated development and eventual enrichment of the corpus of mathematics.

Acknowledgement

I thank J. W. Addison, Robert Boyer, and two anonymous referees for encouragement and many useful comments on earlier drafts of this paper.

Appendix 1. Introduction to Resolution Theorem Proving

A1. INTRODUCTION

I will provide a brief overview of the principal methods used in resolution theorem proving, with particular reference to the system OTTER. This appendix is expository, and no proofs are provided. For further discussion, the reader may consult [21, 10, 5, 9 and 15].

A2. CLAUSES

Many computer implementations of first-order logic use the clausal form, which is an equivalent formulation of this logic. I will use the following language conventions in describing clausal form.

Names are arbitrary strings over the uppercase and lowercase English alphabet. Names beginning with lower case 'u' through 'z' are variables. All other names are either individual constant symbols, function symbols, or relation symbols, depending upon the context.

Terms are defined inductively in the usual way, as the least class such that

(1) A variable or an individual constant symbol is a term.
(2) If f is a function symbol and $t_1, t_2, \dots, t_n$ are terms, then $f(t_1, t_2, \dots, t_n)$ is a term.

If R is a relation symbol and $t_1, t_2, \dots, t_n$ are terms, then $R(t_1, t_2, \dots, t_n)$ is an *atomic formula.*

If $A_1, \dots, A_m, B_1, \dots, B_n$ are atomic formulas, then $A_1 \;\&\dots\&\; A_m \rightarrow B_1 \vee \dots \vee B_n$ is a *clause*. I will always abbreviate this clause as $A_1, \dots, A_m \rightarrow B_1, \dots, B_n$, so that commas appearing before the conditional sign stand for &, while those following the conditional sign stand for $\vee$. I will use the conditional sign '$\rightarrow$' in clauses (sequents), and reserve '$\Rightarrow$' for use in ordinary formulas of first-order logic.

Special cases of clauses:

$m = 0$ (no hypotheses):

$$\rightarrow B_1, \dots, B_n \quad \text{iff} \quad (B_1 \vee \dots \vee B_n)$$

$n = 0$ (no conclusions):

$$A_1, \ldots, A_m \rightarrow \quad \text{iff} \quad \neg(A_1 \,\&\, \ldots \,\&\, A_m)$$

where with $m = 1$, we see how negations are represented.

$m = 0$ and $n = 0$ (no hypotheses or conclusions):

$\rightarrow$ is the null clause, which is *false*.

$n = 0$ or $n = 1$ (Horn clause):

$$A_1, \ldots, A_m \rightarrow [B]$$

which are the rules used in the logic programming language Prolog.

A3. CONVERSION TO CLAUSAL FORM

It is often necessary to convert formulas written in standard first-order notation into clausal form. I have written a Prolog program (using many routines supplied by William McCune) to do this conversion. The principal steps taken by my program are:

(1) Verify that the input formula is well-formed.
(2) Replace the formula by its universal closure.
(3) Eliminate $\Rightarrow$ and $\Leftrightarrow$ in favor of $\neg$, $\vee$ and &.
(4) Drive all negation signs in to the atomic formulas.
(5) Optimize the placement of quantifiers, to reduce the number and arity of the Skolem functions introduced in (6).
(6) Replace existentially quantified variables by Skolem terms.
(7) Delete universal quantifiers.
(8) Place the matrix in conjunctive normal form (a conjunction of clauses).
(9) Delete any clause that is a tautology.
(10) Check whether any clause can be subsumed by a simpler factor.
(11) Delete any clause that is subsumed by another clause.
(12) Write each clause as a sequent.

Every step of this procedure produces output that is logically equivalent to the input – except for step (6), which only preserves satisfiability. Thus the universal closure of the resulting conjunction of clauses is satisfiable iff the universal closure of the original formula is satisfiable.

The skeleton of this procedure may be found in many texts, such as [9]. OTTER also includes a module to carry out most of the above steps.

A4. CUT RULE

We let Greek capitals Γ, Σ, Φ, Θ stand for finite lists of atomic formulas, separated by commas. The *cut rule* in Gentzen systems is the propositional inference rule

$$\frac{\begin{array}{l}\Gamma \to \Sigma, R \\ R, \Phi \to \Theta\end{array}}{\Gamma, \Phi \to \Sigma, \Theta.}$$

This rule is a generalization of the chain rule of inference.

A5. SUBSTITUTIONS

An *expression* is either a term, a list of terms, an atomic formula, or a list of atomic formulas. A finite set $\sigma = \{\langle v_1, t_1\rangle, \ldots, \langle v_n, t_n\rangle\}$ of ordered pairs is a *substitution* iff every v_i is a variable, every t_i is a term, and the variables are distinct. Conventionally 'ε' denotes the null substitution.

If σ is a substitution and E is an expression, then $E\sigma$, the *instance* of E by σ, is the expression obtained from E by simultaneously replacing each occurrence of the variable v_i by t_i, for $i = 1, \ldots, n$.

If $\mu = \{\langle u_1, s_1\rangle, \ldots, \langle u_m, s_m\rangle\}$ is another substitution, then the *composition* $\mu\sigma$ of μ and σ is the substitution obtained from the set

$$\{\langle u_1, s_1\sigma\rangle, \ldots, \langle u_m, s_m\sigma\rangle, \langle v_1, t_1\rangle, \ldots, \langle v_n, t_n\rangle\}$$

by deleting any pair $\langle u_i, s_i\sigma\rangle$ for which $u_i = s_i\sigma$, and deleting any pair $\langle v_j, t_j\rangle$ for which $v_j \in \{u_1, \ldots, u_m\}$.

For any substitutions μ, σ, τ and expression E, we have the following:

$$\sigma\varepsilon = \varepsilon\sigma = \sigma,$$

$$(E\mu)\sigma = E(\mu\sigma),$$

$$(\mu\sigma)\tau = \mu(\sigma\tau).$$

A6. UNIFICATION ALGORITHM

The unification algorithm is a method for finding the most general substitution that will make two expressions identical, if any such substitution exists. If a most general unifier exists, it is unique up to a renaming of variables. The method is similar in spirit to the solution of a system of linear equations by the successive elimination of variables.

We let *unify(E1, E2, σ)* mean that the substitution σ is a most general unifier of the expressions *E1* and *E2*. Computer scientists often specify algorithms in pseudo-Pascal. It is an embarassment that such procedural specifications are even used in works on logic programming! Since I endorse the thesis that computation is controlled deduction, I will instead specify the algorithm for computing *unify* in pseudo-Prolog.

```
unify(VorC, VorC, ε) ←
      VorC is a variable or an individual constant.
unify(V, T, {<V, T>}) ←
      V is a variable, T is a term,
      V does not occur in T.
unify(T, V, {<V, T>}) ←
      V is a variable, T is a term,
      V does not occur in T.
unify(T1, T2, σ) ←
      T1 =.. [Funct | Args1], T2 =.. [Funct | Args2],
      unify_list(Args1, Args2, σ).

unify_list([], [], ε).
unify_list([T1 | T1s], [T2 | T2s], (μ σ)) ←
      unify(T1, T2, μ),
      unify_list((T1s μ), (T2s μ), σ).
```

The principal theorem concerning this algorithm is that it always terminates, and if it terminates in success, the output σ is indeed a most general unifier of *E1* and *E2*. We can even use the above two procedures to find the most general unifier of any finite list $[E_1, \ldots, E_n]$ of unifiable expressions, by calling *unify_list*($[E_1, \ldots, E_{n-1}]$, $[E_2, \ldots, E_n]$, σ).

A7. BINARY RESOLUTION

For each inference rule described below, the premises of the inference are first *standardized apart* by renaming variables so that they have no variable in common.

Combining the cut rule with the unification algorithm, we obtain the *binary resolution* inference rule.

Almost General Case:

$$\frac{\Gamma \to \Sigma, R \qquad R', \Phi \to \Theta}{(\Gamma, \Phi \to \Sigma, \Theta)\,\sigma,}$$

where R and R' are unifiable with most general unifier σ.

In the fully general case, we allow σ to unify R, R', *and other literals* from Σ and Φ, which are also deleted from the conclusion. If we do not permit such additional unifications, then we also must use *factoring*, a separate rule of inference, to obtain refutation completeness.

The intuitive importance of using the unification algorithm while computing resolvents is that it only instantiates variables to the minimum degree necessary, thus keeping deduced information in the most general form possible. The algorithm makes *intelligent* substitutions for variables, as opposed to the blind substitutions used in early theorem provers based upon Herbrand's theorem.

Binary resolution is a *refutation-complete* rule of inference for first-order logic expressed in clausal form. By this we mean that if a set of clauses is unsatisfiable, then by a series of applications of binary resolution to the clauses it is possible to derive the null clause – signifying a contradiction. Thus in the use of binary resolution (or other resolution rules discussed below), we convert the negation of the proposed theorem to clausal form, and attempt to derive the null clause.

I never use binary resolution, since it generates far too much 'junk' to use regularly in automated proofs. Usually the binary resolvent is a longer clause than either of the input clauses. Deriving longer clauses is heading in the wrong direction from trying to derive the null clause, that is from proving the theorem. The reasoning steps taken by binary resolution are too small. Binary resolution deserves a large historical accolade, but today it is of interest mainly as the basis for more powerful procedures such as hyperresolution and UR-resolution.

A8. HYPERRESOLUTION

Special Case: Forward chaining from a Horn clause.

$$\begin{array}{l} A_1, \ldots, A_m \rightarrow B \\ \qquad\quad\;\; \rightarrow A_1' \\ \qquad\qquad \vdots \\ \qquad\quad\;\; \rightarrow A_m' \\ \hline \qquad\quad\;\; \rightarrow B\sigma, \end{array}$$

where σ is the most general unifier making $A_i\sigma = A_i'\sigma$ for all i.

Almost General Case: Non-Horn clause nucleus.

$$\begin{array}{l} A_1, \ldots, A_m \rightarrow \Phi \\ \qquad\quad\;\; \rightarrow \Theta_1, A_1' \\ \qquad\qquad \vdots \\ \qquad\quad\;\; \rightarrow \Theta_m, A_m' \\ \hline \qquad\quad\;\; \rightarrow (\Phi, \Theta_1, \ldots, \Theta_m)\sigma. \end{array}$$

The almost general case is the same as the special case, except for the extra disjuncts that tag along for the ride. Note that every input clause but one is a *positive* clause, and the conclusion is a positive clause.

In the fully general case, just as with binary resolution, we allow σ also to unify other literals from the Θ_is, which are also deleted from the conclusion.

Hyperresolution (like binary resolution) is a refutation-complete rule of inference.

A9. UR-RESOLUTION

The disjunctions produced as output of hyperresolution are waffling conclusions.

Deriving further clauses from such a conclusion means treating several cases simultaneously, when it would usually be more efficient to treat them serially. But OTTER lacks a case-analysis mechanism.

Intuitively speaking, *unit* clauses are the powerhouses of an automated reasoning system. They assert that something definitely *is* (or *is not*) the case. A unit clause is just one literal away from the null clause – proof of the theorem.

UR-resolution (unit-resulting resolution) is to *unit* clauses as hyperresolution is to *positive* clauses.

$$\begin{array}{rl}
A_1, \ldots, A_m & \to B_1, \ldots, B_n \\
 & \to A'_1 \\
 & \vdots \\
 & \to A'_m \\
B'_1 & \to \\
 & \vdots \\
B'_{n-1} & \to \\
\hline
 & \to B_n\sigma,
\end{array}$$

where σ is the most general unifier making $A_i\sigma = A'_i\sigma$ for all i and $B_i\sigma = B'_i\sigma$ for $i \neq n$.

Every literal but one (which, unlike the illustration, does not have to be the last one, and can be positive or negative) is cancelled from the nucleus under a most general unifier σ. So the conclusion is another unit clause.

UR-resolution is refutation complete for a system of assumptions all of which are Horn clauses. (It is a very useful rule in any case.)

Both hyperresolution and UR-resolution are substantial improvements over binary resolution, in that they make larger inference steps in one fell swoop, without saving the intermediate results to further clog up the clause space. They are effective steps in fighting the combinatorial explosion.

A10. EQUALITY REASONING

The equality relation is one of the most important relations used in mathematical reasoning. Its principal logical feature is that one can substitute equals for equals in any expression.

In the standard treatment of equality as an addition to first-order logic, we add reflexivity, symmetry, and transtivity axioms, together with substitutivity axioms for every function symbol and relation symbol in the system. While this treatment also will work in an automated reasoning system, it is *very* slow and inefficient. Using these axioms, several chaining steps are needed to carry out simple substitutions.

Paramodulation

Paramodulation is a clausal equality inference rule that permits us to make equality substitutions directly at the term level.

Let $C[r]$ be a clause containing a term r at some designated position. If t is another term, we let $C[t]$ be the result of replacing this single designated occurrence of r by t. The paramodulation rule of inference is

$$\frac{\begin{array}{ll} C[r] & (\text{'into' clause}) \\ \Gamma \to (s = t), \Sigma & (\text{'from' clause}) \end{array}}{(\Gamma, C[t], \Sigma)\,\sigma,}$$

where σ is the most general unifier of r and s. Note that C carries the sequent sign.

Paramodulation provides a logically complete treatment of equality, without the need to add the special axioms mentioned above, except reflexivity.

Paramodulation, like binary resolution, has the potential to generate large amounts of 'junk' that clogs up the clause space. This is particularly true if one allows the terms r or s to be *variables*. But most of the theorems I prove are universal statements, in which the clausal version of the denial of the theorem contains no free variables. In such cases I also assign a high enough weight to variables so that all deduced clauses containing variables will be discarded. Thus any proof obtained will be a ground proof. This heuristic is quite successful in pruning the output of paramodulation to a manageable set of more useful clauses.

Demodulation

Demodulators (rewrite rules, reductions) are positive unit equality clauses that are used to simplify or canonicalize other expressions. A single demodulation occurs as follows:

$$\frac{\begin{array}{l} C[r] \\ \to (s = t) \end{array}}{C[t\sigma],}$$

where σ is a substitution such that $s\sigma = r$. Note that here we are using one-way matching, rather than unification.

Normally, all demodulators in the system are applied to any newly derived clause until it cannot be further simplifed. The derived clause is then discarded in favor of the simplifed clause. OTTER also provides a *back demodulation* option, wherein newly derived demodulators are used to simplify all clauses already in the system.

Use of equality simplifications is a powerful, fast, and particularly easy to understand reasoning procedure. Knuth and Bendix [8] provided an algorithm for a class of equational theories that permits the computation of a set of rewrite rules sufficient to check the truth of every equation of the theory by requiring that equal terms reduce to the same normal form. Following their seminal work, complete sets of reductions have been found for a number of equational theories (see, for example, Section 6 of this paper).

Lexical Ordering.

A demodulator $\rightarrow (s = t)$ is *lex-dependent* iff s and t become identical when all variables are replaced by the same symbol, say 0. An example of a lex-dependent demodulator is $\rightarrow ((x + y) = (y + x))$, which will loop if applied without restriction. But one can use the *lex* command of OTTER to assign a lexical ordering on function symbols and individual constants. This ordering induces a lexical ordering on all terms. A lex-dependent demodulator is then applied only if it produces a lexically smaller term.

Lexical ordering is also used to determine the order of deduced equality literals. Normally the term of lowest weight is placed on the right side of the equality. If both terms have the same weight, the lexically smallest term is placed on the right side.

A11. OTHER STRATEGIES FOR FIGHTING THE COMBINATORIAL EXPLOSION

I will briefly summarize a few other significant strategies used by resolution theorem provers.

Set-of-Support Strategy

As normally used, this strategy demands that each conclusion drawn be relevant to the particular theorem we are trying to prove, by requiring that at least one of the premise clauses of the inference have a clause from the theorem as an ancestor.

Subsumption

Clause A subsumes clause B if there exists a substitution σ such that $A\sigma \subseteq B$. The subsumption procedure discards any derived clause that is less general than another clause already in the clause space.

Weighting

The default *weight* of a clause is just the sum of the number of individual variables, individual constants, function symbols, and relation symbols it contains; connectives are not counted. One can use weight templates to modify this default computation. For example, the weight template $weight(f(g(2), 3), 50)$ assigns weight $(2 \cdot \text{weight of } r) + (3 \cdot \text{weight of } s) + 50$ to the term $f(g(r), s)$. We normally use weights in two ways: to discard any generated clause with weight beyond a specified maximum, and to select the clause of smallest weight from the set of support as the next clause from which to draw inferences.

A12. RUNNING OTTER

The basic loop carried out by OTTER, as I normally use it, is in pseudo-Prolog:

```
proof ←
    the null clause is derived.
proof ←
    the set of support is empty, !, fail.
proof ←
    the alotted resources are exceeded, !, fail.
proof ←
    let the given clause be the clause of lowest weight in the set of support,
    use the chosen rules of inference to derive all consequences of the given
    clause and the axiom list (the given clause must participate in the inference),
    process the derived clauses by demodulation, subsumption checks, etc.,
    move the given clause to the axiom list, and add the processed derived
    clauses to the set of support,
    proof.
```

Once I prove a theorem, I add it to the axiom list or to the demodulator list so that it may be used in the proof of further theorems.

Appendix 2. Theorems Proved

I only supply the first-order form of the theorem in a few cases where it is not obvious from the clauses. Equality theorems preceded by '%' are not loaded with the axiom list, but rather with the demodulator list. To save space, I have not provided that separate list. The few theorems preceded by '%%' are subsumed by other theorems.

PARTIAL ORDER

(PO1) Reflexivity.
$\rightarrow (x \subseteq x)$.

(PO2) Antisymmetry is part of Axiom A-3.
%%$(x \subseteq y), (y \subseteq x) \rightarrow (x = y)$.

(PO3) Transitivity.
$(x \subseteq y), (y \subseteq z) \rightarrow (x \subseteq z)$.

EQUALITY

(EQ1) Boyer et al.'s equality axiom.
$\forall x(x = x)$.

$\rightarrow (x = x)$.

(EQ2) Expanded equality definition.
$\rightarrow$ *$(x = y)$, $(notsub(x, y) \in x)$, $(notsub(y, x) \in y)$.*
$(notsub(x, y) \in y)$ $\rightarrow$ $(x = y)$, $(notsub(y, x) \in y)$.
$(notsub(y, x) \in x)$ $\rightarrow$ $(x = y)$, $(notsub(x, y) \in x)$.
$(notsub(x, y) \in y)$, $(notsub(y, x) \in x)$ $\rightarrow$ $(x = y)$.

SPECIAL CLASSES

(SP1) Lemma.
$(y \in (x \cap \sim(x)))$ $\rightarrow$.

(SP2) Existence of *0* (null class).
$\exists x\ \forall z(\neg(z \in x))$.

$(z \in 0)$ $\rightarrow$.

(SP3) *0* is a subclass of every class.
$\rightarrow$ *$(0 \subseteq x)$.*
Corollary.
$(x \subseteq 0)$ $\rightarrow$ $(x = 0)$.

(SP4) Uniqueness of null class.
$\rightarrow$ *$(z = 0)$, $(notsub(z, 0) \in z)$.*

(SP5) *0* is a set (follows from axiom of infinity).
$\rightarrow$ *$(0 \in V)$.*

UNORDERED PAIRS

(UP1) Unordered pair is commutative.
$\rightarrow$ *$(\{x, y\} = \{y, x\})$.*

(UP2) If one argument is a proper class, pair contains only the other.
$\rightarrow$ *$(\{x, x\} \subseteq \{x, y\})$.*
$\rightarrow$ *$(y \in V)$, $(\{x, y\} = \{x\})$.*

(UP3) If both arguments are proper classes, pair is null.
$\rightarrow$ *$(\{x, y\} = 0)$, $(x \in V)$, $(y \in V)$.*

(UP4) Left cancellation for unordered pairs.
$(\{x, y\} = \{x, z\})$, $(<y, z> \in (V \times V))$ $\rightarrow$ $(y = z)$.

(UP5) Right cancellation for unordered pairs.
$(\{x, z\} = \{y, z\})$, $(<x, y> \in (V \times V))$ $\rightarrow$ $(x = y)$.

(UP6) Corollary to (A-4).
$(x \in V)$, $(\{x, y\} = 0)$ $\rightarrow$.
$(y \in V)$, $(\{x, y\} = 0)$ $\rightarrow$.

(UP7) If both members of a pair belong to a set, the pair is a subset.
$(x \in z)$, $(y \in z)$ $\rightarrow$ $(\{x, y\} \subseteq z)$.

SINGLETONS

(SS1) Every singleton is a set.
$\rightarrow$ *$(\{x\} \in V)$.*

(SS2) A set belongs to its singleton.
$(x \in V)$ $\rightarrow$ $(x \in \{x\})$.
Corollary.
$(x \in V)$, $(\{x\} = 0)$ $\rightarrow$.

(SS3) Only x can belong to $\{x\}$.
$(y \in \{x\}) \rightarrow (y = x)$.

(SS4) If x is not a set, $\{x\} = 0$.
$\rightarrow (x \in V), (\{x\} = 0)$.

(SS5) A singleton set is determined by its element.
$(\{x\} = \{y\}), (x \in V) \rightarrow (x = y)$.
$(\{x\} = \{y\}), (y \in V) \rightarrow (x = y)$.

(SS6) Existence of *memb*.
$\forall x(\exists u((u \in V \,\&\, x = \{u\}) \vee (\neg \exists y(y \in V \,\&\, x = \{y\}) \,\&\, u = x)))$.

$(y \in V) \rightarrow (memb(\{y\}) \in V)$.
$(y \in V) \rightarrow (\{memb(\{y\})\} = \{y\})$.
$\rightarrow (memb(x) \in V), (memb(x) = x)$.
$\rightarrow (\{memb(x)\} = x), (memb(x) = x)$.

(SS7) Uniqueness of *memb* of a singleton set.
$\forall x \, \forall u(((u \in V) \,\&\, x = \{u\}) \Rightarrow memb(x) = u)$.

$(u \in V) \rightarrow (memb(\{u\}) = u)$.

(SS8) Uniqueness of *memb* when x is not a singleton of a set.
$\forall x \, \forall u((\neg \exists y((y \in V) \,\&\, x = \{y\}) \,\&\, u = x) \Rightarrow memb(x) = u)$.

$\rightarrow (memb'(x) \in V), (memb(x) = x)$.
$\rightarrow (\{memb'(x)\} = x), (memb(x) = x)$.

(SS9) Corollary to (SS1).
$(\{memb(x)\} = x) \rightarrow (x \in V)$.

(SS10).
$(\{memb(x)\} = x), (y \in x) \rightarrow (memb(x) = y)$.

(SS11).
$(x \in y) \rightarrow (\{x\} \subseteq y)$.

(SS12).
$(x \subseteq \{y\}) \rightarrow (\{y\} = x), (x = 0)$.

(SS13) A class contains 0, 1, or at least 2 members.
$\rightarrow (notsub((\sim(\{notsub(x, 0)\}) \cap x), 0) \in (\sim(\{notsub(x, 0)\}) \cap x)), (\{notsub(x, 0)\} = x), (x = 0)$.
Corollaries.
$\rightarrow (notsub((\sim(\{notsub(x, 0)\}) \cap x), 0) \in x), (\{notsub(x, 0)\} = x), (x = 0)$.
$(notsub((\sim(\{notsub(x, 0)\}) \cap x), 0) = notsub(x, 0)) \rightarrow (\{notsub(x, 0)\} = x), (x = 0)$.

ORDERED PAIRS

(OP1) An ordered pair is a set.
$\rightarrow (<x, y> \in V)$.

(OP2) Members of ordered pair.
$\rightarrow (\{x\} \in <x, y>)$.
$\rightarrow (\{x, \{y\}\} \in <x, y>)$.

(OP3) Special cases.
$\rightarrow (\{\{x\}, \{x, 0\}\} = <x, y>), (y \in V), .$
$\rightarrow (\{0, \{\{y\}\}\} = <x, y>), (x \in V)$.
$\rightarrow (\{0, \{0\}\} = <x, y>), (x \in V), (y \in V)$.

(OP4)-(OP5) An ordered pair uniquely determines its components.

(OP4).
%%$(\langle w, x\rangle = \langle y, z\rangle), (w \in V), (y \in V) \rightarrow (w = y)$.

(OP5).
%%$(\langle w, x\rangle = \langle y, z\rangle), (x \in V), (z \in V) \rightarrow (x = z)$.

(OP6) Existence of *1st* and *2nd*.
$\forall x\, \exists u\, \exists v((\langle u, v\rangle \in (V \times V)\ \&\ x = \langle u, v\rangle)\ \mathrm{v}$
$(\neg\, \exists y\, \exists z(\langle y, z\rangle \in (V \times V)\ \&\ x = \langle y, z\rangle)\ \&\ u = x\ \&\ v = x))$.

$(\langle y, z\rangle \in (V \times V)) \rightarrow (\langle 1st(\langle y, z\rangle), 2nd(\langle y, z\rangle)\rangle \in (V \times V))$.
$(\langle y, z\rangle \in (V \times V)) \rightarrow (\langle 1st(\langle y, z\rangle), 2nd(\langle y, z\rangle)\rangle = \langle y, z\rangle)$.
$\rightarrow (\langle 1st(x), 2nd(x)\rangle \in (V \times V)), (1st(x) = x)$.
$\rightarrow (\langle 1st(x), 2nd(x)\rangle \in (V \times V)), (2nd(x) = x)$.
$\rightarrow (\langle 1st(x), 2nd(x)\rangle = x), (1st(x) = x)$.
$\rightarrow (\langle 1st(x), 2nd(x)\rangle = x), (2nd(x) = x)$.

(OP7) Uniqueness of *1st* and *2nd* when *x* is an ordered pair of sets.
$\forall x\, \forall u\, \forall v((\langle u, v\rangle \in (V \times V)\ \&\ x = \langle u, v\rangle) \Rightarrow 1st(x) = u\ \&\ 2nd(x) = v)$

$(\langle u, v\rangle \in (V \times V)) \rightarrow (1st(\langle u, v\rangle) = u)$.
$(\langle u, v\rangle \in (V \times V)) \rightarrow (2nd(\langle u, v\rangle) = v)$.

(OP8) Uniqueness of *1st* and *2nd* when *x* is not an ordered pair of sets.
$\forall x\, \forall u\, \forall v((\exists y\, \exists z((\langle y, z\rangle \in (V \times V))\ \&\ x = \langle y, z\rangle)\ \&\ u = x\ \&\ v = x)\ \mathrm{v}\ 1st(x) = u\ \&\ 2nd(x) = v)$.

$\rightarrow (\langle 1st'(x), 2nd'(x)\rangle \in (V \times V)), (1st(x) = x)$.
$\rightarrow (\langle 1st'(x), 2nd'(x)\rangle \in (V \times V)), (2nd(x) = x)$.
$\rightarrow (\langle 1st'(x), 2nd'(x)\rangle = x), (1st(x) = x)$.
$\rightarrow (\langle 1st'(x), 2nd'(x)\rangle = x), (2nd(x) = x)$.

(OP9) Corollaries to (OP1).
$(\langle 1st(x), 2nd(x)\rangle = x) \rightarrow (x \in V)$.

(OP10) Improved version of (OP4).
$(\langle w, x\rangle = \langle y, z\rangle), (w \in V) \rightarrow (w = y)$.
Corollaries.
$\rightarrow (x \in V), (1st(\langle x, y\rangle) = \langle x, y\rangle)$.
$\rightarrow (x \in V), (2nd(\langle x, y\rangle) = \langle x, y\rangle)$.

(OP11) Improved version of (OP5).
$(\langle w, x\rangle = \langle y, z\rangle), (x \in V) \rightarrow (x = z)$.
Corollaries.
$\rightarrow (y \in V), (1st(\langle x, y\rangle) = \langle x, y\rangle)$.
$\rightarrow (y \in V), (2nd(\langle x, y\rangle) = \langle x, y\rangle)$.

BOOLEAN ALGEBRA

INTERSECTION

(I1) Associative law of intersection.
$\rightarrow (((x \cap y) \cap z) = (x \cap (y \cap z)))$.

(I2) Commutative law of intersection.
%$\rightarrow ((x \cap y) = (y \cap x))$.

(I3) Lexical ordering within associations.
%$\rightarrow ((x \cap (y \cap z)) = (y \cap (x \cap z)))$.

(I4) Intersection with *0*.
%$\rightarrow ((0 \cap x) = 0)$.

(I5) V is an identity for intersection.
$\% \rightarrow ((V \cap x) = x).$

(I6); Idempotent law of intersection.
$\% \rightarrow ((x \cap x) = x).$
Corollary.
$\% \rightarrow ((x \cap (x \cap y)) = (x \cap y)).$

COMPLEMENT

(C1) Complement of complement.
$\% \rightarrow (\sim(\sim(x)) = x).$

(C2) Special cases.
$\% \rightarrow (\sim(0) = V).$
$\% \rightarrow (\sim(V) = 0).$

(C3) Intersection and union with complement.
$\% \rightarrow ((\sim(x) \cap x) = 0).$
$\% \rightarrow ((\sim(x) \cup x) = V).$

(C4) DeMorgan's laws.
$\% \rightarrow (\sim((x \cup y)) = (\sim(x) \cap \sim(y))).$
$\% \rightarrow (\sim((x \cap y)) = (\sim(x) \cup \sim(y))).$

(C5) Uniqueness of complement.
$((x \cup y) = V), ((x \cap y) = 0) \rightarrow (\sim(x) = y).$

UNION

(U1) Associative law of union.
$\rightarrow (((x \cup y) \cup z) = (x \cup (y \cup z))).$

(U2) Commutative law of union
$\% \rightarrow ((x \cup y) = (y \cup x)).$

(U3) Lexical ordering within associations.
$\% \rightarrow ((x \cup (y \cup z)) = (y \cup (x \cup z))).$

(U4) 0 is identity for union.
$\% \rightarrow ((0 \cup x) = x).$

(U5) Union with V.
$\% \rightarrow ((V \cup x) = V).$

(U6) Idempotent law of union.
$\% \rightarrow ((x \cup x) = x).$
Corollary.
$\% \rightarrow ((x \cup (x \cup y)) = (x \cup y)).$

(U7) Members of union.
$(x \in (y \cup z)) \rightarrow (x \in y), (x \in z).$
$(x \in y) \rightarrow (x \in (y \cup z)).$
$(x \in z) \rightarrow (x \in (y \cup z)).$

DISTRIBUTIVE LAWS

(D1) Intersection distributes over union.
$\% \rightarrow (((x \cap y) \cup (x \cap z)) = (x \cap (y \cup z))).$
$\% \rightarrow (((x \cap z) \cup (y \cap z)) = ((x \cup y) \cap z)).$

(D2) Union distributes over intersection.
$\rightarrow (((x \cup y) \cap (x \cup z)) = (x \cup (y \cap z)))$.
$\rightarrow (((x \cup z) \cap (y \cup z)) = ((x \cap y) \cap z))$.

(D3) Absorption for intersection.
$\rightarrow ((x \cap (x \cup y)) = x)$.
Corollary.
$\rightarrow ((x \cap (y \cap (x \cup z))) = (x \cap y))$.

(D4) Absorption for union.
$\rightarrow ((x \cup (x \cap y)) = x)$.
Corollary.
$\rightarrow ((x \cup (y \cup (x \cap z))) = (x \cup y))$.

(D5).
$\rightarrow ((x \cup (\sim(x) \cap z)) = (x \cup z))$.
Corollary.
$\rightarrow ((x \cup (y \cup (\sim(x) \cap z))) = (x \cup (y \cup z)))$.
$\rightarrow (((x \cap z) \cup (\sim(x) \cap (y \cap z))) = ((x \cap z) \cup (y \cap z)))$.

(D6).
$\rightarrow ((\sim(x) \cup (x \cap z)) = (\sim(x) \cup z))$.
Corollary.
$\rightarrow ((\sim(x) \cup (y \cup (x \cap z))) = (\sim(x) \cup (y \cup z)))$.

(D7).
$\rightarrow (((\sim(x) \cap y) \cup (x \cap y)) = y)$.

SUBCLASSES

(SU1).
$(x \subseteq y) \rightarrow ((x \cap y) = x)$.

(SU2).
$((x \cap y) = x) \rightarrow (x \subseteq y)$.

(SU3).
$(x \subseteq y) \rightarrow ((x \cup y) = y)$.

(SU4).
$((x \cup y) = y) \rightarrow (x \subseteq y)$.

(SU5).
$(x \subseteq y) \rightarrow ((\sim(y) \cap x) = 0)$.

(SU6).
$((\sim(y) \cap x) = 0) \rightarrow (x \subseteq y)$.

(SU7).
$(x \subseteq y) \rightarrow ((\sim(x) \cup y) = V)$.

(SU8).
$((\sim(x) \cup y) = V) \rightarrow (x \subseteq y)$.

(SU9).
$(x \subseteq y) \rightarrow (\sim(y) \subseteq \sim(x))$.

(SU10).
$(\sim(y) \subseteq \sim(x)) \rightarrow (x \subseteq y)$.

LATTICE

(LA1) Upper and lower bounds.
$\rightarrow (x \subseteq (x \cup y))$.
$\rightarrow (y \subseteq (x \cup y))$.
$\rightarrow ((x \cap y) \subseteq x)$.
$\rightarrow ((x \cap y) \subseteq y)$.

(LA2) Least upper and greatest lower bounds.
$(x \subseteq z), (y \subseteq z) \rightarrow ((x \cup y) \subseteq z)$.
$(z \subseteq x), (z \subseteq y) \rightarrow (z \subseteq (x \cap y))$.

(LA3) Union and intersection are monotonic.
$(x \subseteq y) \rightarrow ((x \cup z) \subseteq (y \cup z))$.
$(x \subseteq y) \rightarrow ((x \cap z) \subseteq (y \cap z))$.

CARTESIAN PRODUCT

(CP1).
$\rightarrow ((x \times y) \subseteq (V \times V))$.
Corollary.
$(u \in x), (v \in y) \rightarrow (<u, v> \in (V \times V))$.

(CP2).
$(<u, v> \in (x \times y)) \rightarrow (<v, u> \in (y \times x))$.

(CP3) Special cases.
$\% \rightarrow ((x \times 0) = 0)$.
$\% \rightarrow ((0 \times y) = 0)$.

(CP4).
$\rightarrow ((x \cap (V \times V)) \subseteq (D(x) \times V))$.

(CP5) $\times$ is monotonic.
$(x \subseteq y) \rightarrow ((x \times z) \subseteq (y \times z))$.
$(y \subseteq z) \rightarrow ((x \times y) \subseteq (x \times z))$.
Corollaries.
$\rightarrow ((x \times z) \subseteq ((x \cup y) \times z))$.
$\rightarrow ((y \times z) \subseteq ((x \cup y) \times z))$.
$\rightarrow ((x \times y) \subseteq (x \times (y \cup z)))$.
$\rightarrow ((x \times z) \subseteq (x \times (y \cup z)))$.
$\rightarrow (((x \cap y) \times z) \subseteq (x \times z))$.
$\rightarrow (((x \cap y) \times z) \subseteq (y \times z))$.
$\rightarrow ((x \times (y \cap z)) \subseteq (x \times y))$.
$\rightarrow ((x \times (y \cap z)) \subseteq (x \times z))$.

(CP6) $\times$ distributes over union.
$\% \rightarrow (((x \times z) \cup (y \times z)) = ((x \cup y) \times z))$.
$\% \rightarrow (((x \times y) \cup (x \times z)) = (x \times (y \cup z)))$.

(CP7) $\times$ distributes over intersection.
Special case of (CP9).
$\% \rightarrow (((x \times z) \cap (y \times z)) = ((x \cap y) \times z))$.
$\% \rightarrow (((x \times y) \cap (x \times z)) = (x \times (y \cap z)))$.

(CP8) Lemma.
$\rightarrow (((w \times x) \cap (y \times z)) \subseteq (w \times z))$.

(CP9) Double distribution for intersection.
$\% \rightarrow (((w \times x) \cap (y \times z)) = ((w \cap y) \times (x \cap z)))$.

(CP10) Inverse of square.
$\% \rightarrow (inverse((x \times x)) = (x \times x))$.

(CP10) Left cancellation law.
$((u \times v) = (w \times x)) \rightarrow (u = 0), (v = x)$.
$((u \times v) = (w \times x)) \rightarrow (w = 0), (v = x)$.

(CP11) Right cancellation law.
$((u \times v) = (w \times x)) \rightarrow (v = 0), (u = w)$.
$((u \times v) = (w \times x)) \rightarrow (x = 0), (u = w)$.

(CP12) Corollary.
$((u \times u) = (w \times w)) \rightarrow (u = w)$.

RESTRICTION

(RS1)-(RS4) Alternate definition of *restrict*.

(RS1).
$(\langle u, v\rangle \in restrict(xr, x, y)) \rightarrow (\langle u, v\rangle \in xr)$.

(RS2).
$(\langle u, v\rangle \in restrict(xr, x, y)) \rightarrow (u \in x)$.

(RS3).
$(\langle u, v\rangle \in restrict(xr, x, y)) \rightarrow (v \in y)$.

(RS4).
$(\langle u, v\rangle \in xr), (\langle u, v\rangle \in (x \times y)) \rightarrow (\langle u, v\rangle \in restrict(xr, x, y))$.

(RS5).
$\% \rightarrow (restrict(restrict(xf, x1, y1), x2, y2) = restrict(xf, (x1 \cap x2), (y1 \cap y2)))$.

(RS6) Special cases.
$\% \rightarrow (restrict(V, x, y) = (x \times y))$.
$\% \rightarrow (restrict(0, x, y) = 0)$.
$\% \rightarrow (restrict(xr, 0, y) = 0)$.
$\% \rightarrow (restrict(xr, x, 0) = 0)$.

(RS7) *restrict* preserves intersections.
$\rightarrow ((restrict(xr1, x1, y1) \cap restrict(xr2, x2, y2)) = restrict((xr1 \cap xr2), (x1 \cap x2), (y1 \cap y2)))$.

(RS8).
$\rightarrow ((restrict(xr1, x, y) \cup restrict(xr2, x, y)) = restrict((xr1 \cup xr2), x, y))$.
Corollary.
$\rightarrow ((restrict(x, y, y) \cup inverse(restrict(x, y, y))) = restrict((x \cup inverse(x)), y, y))$.

(RS9) Restriction of E.
$(y \in V) \rightarrow (restrict(E, x, \{y\}) = (x \cap y))$.
$\rightarrow (restrict(x, y, z) \subseteq x)$.
$\rightarrow (restrict(x, y, z) \subseteq (y \times z))$.

(RS10).
$\rightarrow (restrict(x, y, z) \subseteq x)$.
$\rightarrow (restrict(x, y, z) \subseteq (y \times z))$.

(RS11) *restrict* is monotonic.
$(x1 \subseteq x2) \rightarrow (restrict(x1, y, z) \subseteq restrict(x2, y, z))$.
$(y1 \subseteq y2) \rightarrow (restrict(x, y1, z) \subseteq restrict(x, y2, z))$.
$(z1 \subseteq z2) \rightarrow (restrict(x, y, z1) \subseteq restrict(x, y, z2))$.

(RS12).
$\% \rightarrow (restrict((x \times y), x, y) = (x \times y))$.

DOMAIN

(DO1) Alternate version of Axiom B-4.
$\forall z\ \forall xr((z \in D(xr)) \iff (z \in V)\ \&\ \exists y((y \in V)\ \&\ (<z, y> \in xr)))$.

$(x \in D(xr)) \rightarrow (<x, ran(xr, x, V)> \in (V \times V))$.
$(x \in D(xr)) \rightarrow (<x, ran(xr, x, V)> \in xr)$.
$(<x, y> \in xr), (<x, y> \in (V \times V)) \rightarrow (x \in D(xr))$.

(DO2) Special cases.
$\% \rightarrow (D(0) = 0)$.
$\% \rightarrow (D(V) = V)$.

(DO3) Domain preserves union.
$\rightarrow ((D(x) \cup D(y)) = D((x \cup y)))$.
Corollary: domain is monotonic.
$(x \subseteq y) \rightarrow (D(x) \subseteq D(y))$.
$\rightarrow (D((x \cap y)) \subseteq D(x))$.
$\rightarrow (D((x \cap y)) \subseteq D(y))$.

(DO4).
$\rightarrow ((x \cap (V \times V)) \subseteq (D(x) \times V))$.

(DO5) Domain only considers ordered pairs.
$\% \rightarrow (D((x \cap (V \times V))) = D(x))$.

(DO6).
$\rightarrow (D((x \times y)) = x), (y = 0)$.
$\% \rightarrow (D((x \times x)) = x)$.

(DO7).
$\% \rightarrow (restrict(xr, (D(xr) \cap x), y) = restrict(xr, x, y))$.
Corollary.
$\% \rightarrow (restrict(xr, D(xr), y) = restrict(xr, V, y))$.

(DO8).
$\% \rightarrow (D(restrict(x, y, V)) = (D(x) \cap y))$.
Corollaries.
$\rightarrow (D(restrict(x, y, z)) \subseteq (D(x) \cap y))$.
$\rightarrow ((D(restrict(x, y, z)) \times u) \subseteq ((D(x) \cap y) \times u))$.
$\rightarrow ((u \times D(restrict(x, y, z))) \subseteq (u \times (D(x) \cap y)))$.
$\rightarrow ((D(restrict(x1, y1, z1)) \times D(restrict(x2, y2, z2))) \subseteq (y1 \times y2))$.

INVERSE

(IN1)-(IN3) Proof of Gödel's Axiom B-6.
$\forall x(inverse(x) \subseteq (V \times V))$.
$\forall u\ \forall v\ \forall x((<u, v> \in inverse(x)) \iff (<u, v> \in (V \times V))\ \&\ (<v, u> \in x))$.

(IN1).
$\rightarrow (inverse(x) \subseteq (V \times V))$.

(IN2).
$(<u, v> \in inverse(x)) \rightarrow (<v, u> \in x)$.

(IN3).
$(<v, u> \in x), (<u, v> \in (V \times V)) \rightarrow (<u, v> \in inverse(x))$.

(IN4) Special cases.
$\% \rightarrow (inverse(0) = 0)$.
$\% \rightarrow (inverse(V) = (V \times V))$.

(IN5) Inverse distributes over union and intersection.
% → $((inverse(x) \cup inverse(y)) = inverse((x \cup y)))$.
% → $((inverse(x) \cap inverse(y)) = inverse((x \cap y)))$.

(IN6) Domain and range of inverse.
% → $(D(inverse(x)) = R(x))$.
% → $(R(inverse(x)) = D(x))$.

(IN7) Inverse of complement.
→ $(inverse(\sim(x)) = (\sim(inverse(x)) \cap (V \times V)))$.

(IN8) Inverse of product.
% → $(inverse((x \times y)) = (y \times x))$.

(IN9) Inverse of inverse.
% → $(inverse(inverse(x)) = restrict(x, V, V))$.

(IN10) Inverse commutes with restrict.
→ $(restrict(inverse(xr), x, y) = inverse(restrict(xr, y, x)))$.

RANGE

(RA1) Alternate definition of range.
$(y \in R(z))$ → $(<dom(z, V, y), y> \in (V \times V))$.
$(y \in R(z))$ → $(<dom(z, V, y), y> \in z)$.
$(<x, y> \in z), (<x, y> \in (V \times V))$ → $(y \in R(z))$.

(RA2) Special cases.
% → $(R(0) = 0)$.
% → $(R(V) = V)$.

(RA3) Range preserves union.
→ $((R(x) \cup R(y)) = R((x \cup y)))$.

Corollary: range is monotonic.
$(x \subseteq y)$ → $(R(x) \subseteq R(y))$.
→ $(R((x \cap y)) \subseteq R(x))$.
→ $(R((x \cap y)) \subseteq R(y))$.

(RA4).
→ $((x \cap (V \times V)) \subseteq (V \times R(x)))$.

(RA5) Range only considers ordered pairs.
% → $(R((x \cap (V \times V))) = R(x))$.

(RA6).
→ $(R((x \times y)) = y), (x = 0)$.

(RA7).
% → $(restrict(xr, x, (R(xr) \cap y)) = restrict(xr, x, y))$.
Corollary.
% → $(restrict(xr, x, R(xr)) = restrict(xr, x, V))$.

(RA8).
% → $(R(restrict(x, V, z)) = (R(x) \cap z))$.
Corollaries.
→ $(R(restrict(x, y, z)) \subseteq (R(x) \cap z))$.
→ $((R(restrict(x, y, z)) \times u) \subseteq ((R(x) \cap z) \times u))$.
→ $((u \times R(restrict(x, y, z))) \subseteq (u \times (R(x) \cap z)))$.
→ $((R(restrict(x1, y1, z1)) \times R(restrict(x2, y2, z2))) \subseteq (z1 \times z2))$.

(RA9).
$(y \in R(z)) \rightarrow (dom(z, V, y) \in D(z))$.
$(x \in D(z)) \rightarrow (ran(z, x, V) \in R(z))$.

IMAGE

(IM1)-(IM4) Alternate definition of image.

(IM1).
$(y \in (xr \text{ “ } x)) \rightarrow (dom(xr, x, y) \in x)$.
Corollary.
$(y \in (xr \text{ “ } \{x\})), (x \in V) \rightarrow (<x, y> \in xr)$.

(IM2).
$(<x, y> \in xr), (<x, y> \in (V \times V)), (x \in z) \rightarrow (y \in (xr \text{ “ } z))$.
Corollary.
$(<x, y> \in xr), (<x, y> \in (V \times V)) \rightarrow (y \in (xr \text{ “ } \{x\}))$.

(IM3).
$(y \in (xr \text{ “ } x)) \rightarrow (<dom(xr, x, y), y> \in (V \times V))$.

(IM4).
$(y \in (xr \text{ “ } x)) \rightarrow (<dom(xr, x, y), y> \in xr)$.

(IM5) Range is image of the domain.
$\% \rightarrow ((xr \text{ “ } D(xr)) = R(xr))$.
Corollary.
$\% \rightarrow ((xr \text{ “ } V) = R(xr))$.

(IM6) Image is monotonic.
$(y \subseteq z) \rightarrow ((xr \text{ “ } y) \subseteq (xr \text{ “ } z))$.
$(xr \subseteq yr) \rightarrow ((xr \text{ “ } z) \subseteq (yr \text{ “ } z))$.

(IM7).
$((x \cap D(z)) = 0) \rightarrow ((z \text{ “ } x) = 0)$.
Corollaries.
$\% \rightarrow ((z \text{ “ } 0) = 0)$.
$\rightarrow (x \in D(z)), ((z \text{ “ } \{x\}) = 0)$.

(IM8)-(IM9) Alternate definition of subset.

(IM8).
$(x \in \sim((E \text{ “ } \sim(y)))) \rightarrow (x \subseteq y)$.

(IM9).
$(x \subseteq y), (x \in V) \rightarrow (x \in \sim((E \text{ “ } \sim(y))))$.

(IM10) Image under V.
$\rightarrow ((V \text{ “ } x) = V), (x = 0)$.

SUM CLASS

(SC1)-(SC2) Alternate definition of sum class.

(SC1).
$(x \in U(y)) \rightarrow (x \in ran(E, x, y))$.
$(x \in U(y)) \rightarrow (ran(E, x, y) \in y)$.

(SC2).
$(z \in y), (y \in x) \rightarrow (z \in U(x))$.

(SC3) Special cases.
$\% \rightarrow (U(0) = 0)$.
$\% \rightarrow (U(V) = V)$.
$\% \rightarrow (U(\{0\}) = 0)$.

(SC4) Sum of singleton.
$(x \in V) \rightarrow (U(\{x\}) = x)$.

(SC5) Sum of pair.
$(<x, y> \in (V \times V)) \rightarrow (U(\{x, y\}) = (x \cup y))$.
Corollary.
$(<x, y> \in (V \times V)) \rightarrow ((x \cup y) \in V)$.

(SC6) Sum of ordered pair.
$(<x, y> \in (V \times V)) \rightarrow (U(<x, y>) = \{x, \{y\}\})$.

(SC7) An element of *y* is a subset of its union.
$(x \in y) \rightarrow (x \subseteq U(y))$.
Corollary.
$\rightarrow (y \subseteq P(U(y)))$.

(SC8) Alternate definition of sum class.
$\% \rightarrow ((inverse(E) \cdot x) = U(x))$.

(SC9) Sum distributes over union.
$\rightarrow (U((x \cup y)) = (U(x) \cup U(y)))$.

(SC10).
$\rightarrow (U((x \cap y)) \subseteq (U(x) \cap U(y)))$.

(SC11) Domain and range.
$\rightarrow (D(x) \subseteq U(U(x)))$.
$\rightarrow (R(x) \subseteq U(U(x)))$.

POWER CLASS

(PC1)-(PC2) Alternative definition of power class.

(PC1).
$(z \in P(x)) \rightarrow (z \subseteq x)$.

(PC2).
$(z \subseteq x), (z \in V) \rightarrow (z \in P(x))$.

(PC3) Power is monotonic.
$(x \subseteq y) \rightarrow (P(x) \subseteq P(y))$.

(PC4) Special cases.
$\rightarrow (0 \in P(x))$.
$(P(x) = 0) \rightarrow$.
$\% \rightarrow (P(V) = V)$.

(PC5) Power class of a set.
$(x \in V) \rightarrow ((inverse(S) \cdot \{x\}) = P(x))$.

(PC6).
$\rightarrow ((x \times y) \subseteq P(P((x \cup y))))$.

(PC7).
$\rightarrow (x \subseteq U(P(x)))$.

(PC8).
$\rightarrow ((P(x) \cap P(y)) = P((x \cap y)))$.

RELATIONS

(RL1).
$\rightarrow (\mathit{restrict}(xf, x, y) \subseteq (V \times V)).$

(RL2).
$(x \subseteq (V \times V)) \rightarrow (x \subseteq (D(x) \times R(x))).$
Corollaries.
$(x \subseteq (V \times V)) \rightarrow (x \subseteq (D(x) \times (x \text{“} D(x)))).$
$(x \subseteq (V \times V)) \rightarrow (\mathit{restrict}(x, D(x), R(x)) = x).$

(RL3).
$(x \subseteq (V \times V)) \rightarrow (\mathit{inverse}(\mathit{inverse}(x)) = x).$
Corollaries.
$\% \rightarrow (\mathit{inverse}(\mathit{inverse}(\mathit{inverse}(x))) = \mathit{inverse}(x)).$
$\% \rightarrow (\mathit{inverse}(\mathit{inverse}(E)) = E).$
$\% \rightarrow (\mathit{inverse}(\mathit{inverse}(I)) = I).$

COMPOSITION

(CO1)-(CO2) Alternate definition of composition.

(CO1).
$(<u, v> \in (xf \circ yf)) \rightarrow (<u, \mathit{dom}(xf, (yf \text{“} \{u\}), v)> \in yf).$
$(<u, v> \in (xf \circ yf)) \rightarrow (<u, \mathit{dom}(xf, (yf \text{“} \{u\}), v)> \in (V \times V)).$
$(<u, v> \in (xf \circ yf)) \rightarrow (<\mathit{dom}(xf, (yf \text{“} \{u\}), v), v> \in xf).$
$(<u, v> \in (xf \circ yf)) \rightarrow (<\mathit{dom}(xf, (yf \text{“} \{u\}), v), v> \in (V \times V)).$

(CO2).
$(<x, y> \in xr), (<y, z> \in yr), (<x, y> \in (V \times V)), (<y, z> \in (V \times V)) \rightarrow (<x, z> \in (yr \circ xr)).$

(CO3) I is identity for composition.
$\% \rightarrow ((x \circ I) = x).$
$\% \rightarrow ((I \circ x) = x).$

(CO4).
$\rightarrow (\mathit{restrict}(I, D(x), V) \subseteq (\mathit{inverse}(x) \circ x)).$

(CO5) Relation to image.
$\% \rightarrow (((xr \circ yr) \text{“} z) = (xr \text{“} (yr \text{“} z))).$

(CO6) Domain and range of composition.
$\rightarrow (D((xr \circ yr)) \subseteq D(yr)).$
$\rightarrow (R((xr \circ yr)) \subseteq R(xr)).$

(CO7) Composition is associative.
$\rightarrow (((xr \circ yr) \circ zr) = (xr \circ (yr \circ zr))).$

(CO8) Special cases.
$\% \rightarrow ((0 \circ x) = 0).$
$\% \rightarrow ((x \circ 0) = 0).$
$\% \rightarrow ((V \circ x) = (V \times D(x))).$
$\% \rightarrow ((x \circ V) = (R(x) \times V)).$

(CO9) Boyer Lemma 18.
$(R(xr) \subseteq D(yr)) \rightarrow (D((yr \circ xr)) = D(xr)).$

(CO10) Dual version of (CO9).
$(D(yr) \subseteq R(xr)) \rightarrow (R((yr \circ xr)) = R(yr)).$

(CO11) Composition is monotonic.
$(xr \subseteq yr) \rightarrow ((zr \circ xr) \subseteq (zr \circ yr)).$
$(xr \subseteq yr) \rightarrow ((xr \circ zr) \subseteq (yr \circ zr)).$

Corollaries.
$(xr \subseteq yr) \rightarrow ((zr \circ (xr \circ ur)) \subseteq (zr \circ (yr \circ ur)))$.
$(xr \subseteq I) \rightarrow ((zr \circ (xr \circ ur)) \subseteq (zr \circ ur))$.

(CO12) Inverse of composition.
$\rightarrow (inverse((xr \circ yr)) = (inverse(yr) \circ inverse(xr)))$.

(CO13) Composition of element relation.
$(\langle x, y\rangle \in (E \circ E)) \rightarrow (x \in U(y))$.
$(x \in U(y)), (y \in V) \rightarrow (\langle x, y\rangle \in (E \circ E))$.

(CO14DEF) Definition of singleton relation.
$\rightarrow ((\sim((E \circ \sim(I))) \cap E) = SS)$.

(CO15)-(CO17) Membership conditions for SS.

(CO15).
$(\langle x, y\rangle \in SS) \rightarrow (x \in V)$.

(CO16).
$(\langle x, y\rangle \in SS) \rightarrow (\{x\} = y)$.

(CO17).
$(\{x\} = y), (x \in V) \rightarrow (\langle x, y\rangle \in SS)$.

SINGLE-VALUED CLASSES

(SV1)-(SV3) Alternate definition of SINGVAL.

(SV1).
$SINGVAL(z), (\langle u, v\rangle \in (V \times V)), (\langle u, w\rangle \in (V \times V)), (\langle u, v\rangle \in z), (\langle u, w\rangle \in z) \rightarrow (v = w)$.

(SV2DEF) Definitions of terms for (SV3).
$\rightarrow (1st(notsub((x \circ inverse(x)), I)) = sv1(x))$.
$\rightarrow (2nd(notsub((x \circ inverse(x)), I)) = sv2(x))$.
$\rightarrow (dom(x, (inverse(x) \cdot \{sv1(x)\}), sv2(x)) = sv3(x))$.

(SV3).
$\rightarrow (\langle sv3(x), sv1(x)\rangle \in x), SINGVAL(x)$.
$\rightarrow (\langle sv3(x), sv2(x)\rangle \in x), SINGVAL(x)$.
$(sv1(x) = sv2(x)) \rightarrow SINGVAL(x)$.

(SV4) A subclass of a single-valued class is single-valued.
$SINGVAL(x) \rightarrow SINGVAL((x \cap y))$.

(SV5) In a single-valued class, the image of each domain element is a singleton.
$SINGVAL(x), (z \in D(x)) \rightarrow (\{memb((x \cdot \{z\}))\} = (x \cdot \{z\}))$.

(SV6) The composition of single-valued classes is single-valued.
$SINGVAL(xr), SINGVAL(yr) \rightarrow SINGVAL((xr \circ yr))$.

FUNCTIONS

(FU1) The restriction of a function is a function.
$FUNCTION(xf) \rightarrow FUNCTION(restrict(xf, x, y))$.

(FU2) The intersection of functions is a function.
$FUNCTION(xf), FUNCTION(yf) \rightarrow FUNCTION((xf \cap yf))$.

(FU3) The composition of functions is a function.
$FUNCTION(xf), FUNCTION(yf) \rightarrow FUNCTION((xf \circ yf))$.

(FU4) Restriction of function.
$FUNCTION(xf) \rightarrow (restrict(xf, V, V) = xf).$

SUBSET RELATION

(SR1)-(SR3) Alternate definition of S.

(SR1).
$\rightarrow (S \subseteq (V \times V)).$

(SR2).
$(<x, y> \in S) \rightarrow (x \subseteq y).$

(SR3).
$(x \subseteq y), (<x, y> \in (V \times V)) \rightarrow (<x, y> \in S).$

IDENTITY

(ID1)-(ID3) Alternate definition of I.

(ID1).
$\rightarrow (I \subseteq (V \times V)).$

(ID2).
$(<x, y> \in I) \rightarrow (x = y).$

(ID3).
$(x \in V) \rightarrow (<x, x> \in I).$

(ID4) Identity is a function.
$\rightarrow FUNCTION(I).$
Corollary.
$\rightarrow FUNCTION(restrict(I, x, y)).$

(ID5) Domain and range of identity.
$\% \rightarrow (D(I) = V).$
$\% \rightarrow (R(I) = V).$
$\% \rightarrow (D(restrict(I, x, y)) = (x \cap y)).$
$\% \rightarrow (R(restrict(I, x, y)) = (x \cap y)).$
Corollary.
$\% \rightarrow ((I \text{ “ } x) = x).$

(ID6) Image of a class under identity.
$\% \rightarrow ((restrict(I, x, x) \text{ “ } y) = (x \cap y)).$

(ID7) Identity is one-one.
$\rightarrow ONEONE(I).$

(ID8) Inverse of identity is identity.
$\% \rightarrow (inverse(I) = I).$

(ID9) Sets with at most one member.
$((x \times x) \subseteq I) \rightarrow (\{memb(x)\} = x), (x = 0).$
$(x = 0) \rightarrow ((x \times x) \subseteq I).$
$(\{memb(x)\} = x) \rightarrow ((x \times x) \subseteq I).$

(ID10) Sets with more than one member.
$(u \in x) \rightarrow (notsub((x \cap \sim(\{u\})), 0) \in x), ((x \times x) \subseteq I).$
$(u \in x), (notsub((x \cap \sim(\{u\})), 0) = x) \rightarrow ((x \times x) \subseteq I).$

(ID11).
$\rightarrow ((x \cap I) \subseteq (x \cap inverse(x))).$

REPLACEMENT

(RP1) Axiom of Subsets.
$(y \in V), (x \subseteq y) \rightarrow (x \in V).$
$(y \in V) \rightarrow ((x \cap y) \in V).$

(RP2).
$(x \in V) \rightarrow (D(x) \in V).$
$(x \in V) \rightarrow (R(x) \in V).$

(RP3).
$(<x, y> \in (V \times V)) \rightarrow ((x \times y) \in V).$

(RP4).
$(<D(x), R(x)> \in (V \times V)), (x \subseteq (V \times V)) \rightarrow (x \in V).$

(RP5).
$(x \in V) \rightarrow (inverse(x) \in V).$

(RP6).
$FUNCTION(xf), (D(xf) \in V) \rightarrow (xf \in V).$

(RP7).
$SINGVAL(x), (y \in V) \rightarrow (restrict(x, y, V) \in V).$

(RP8).
$(U(x) \in V) \rightarrow (x \in V).$

(RP9).
$(P(x) \in V) \rightarrow (x \in V).$

(RP10).
$((x \cup y) \in V) \rightarrow (x \in V).$

(RP11).
$((x \times y) \in V) \rightarrow (x \in V), (y = O).$
$((x \times y) \in V) \rightarrow (y \in V), (x = O).$

DIAGONALIZATION.

(DI1) Lemma.
$(<x, x> \in y) \rightarrow (x \in D((y \cap I))).$
$(x \in D((y \cap I))) \rightarrow (<x, x> \in y).$

(DI2) Alternate definition of diagonalization.
$\rightarrow (D((\sim(x) \cap y)) = diag(x)).$

(DI3)-(DI4) Alternate definition of diagonalization.

(DI3).
$(z \in diag(xr)), (<z, z> \in xr) \rightarrow .$

(DI4).
$(z \in V) \rightarrow (z \in diag(xr)), (<z, z> \in xr).$

(DI5)-(DI6) Special case of the Russell class, NOT using the axiom of regularity.

(DI5).
$(z \in diag(E)), (z \in z) \rightarrow .$

(DI6).
$(z \in V) \rightarrow (z \in diag(E)), (z \in z).$

(DI7) The Russell class is not a set.
(diag(E) $\in$ V) $\rightarrow$.

(DI8).
$\rightarrow$ *(~(R((xr $\cap$ I))) = diag(xr)).*

(DI9).
$\rightarrow$ *(diag((inverse(xr) $\circ$ xs)) = ~(D((xr $\cap$ xs)))).*
$\rightarrow$ *(diag((xr $\circ$ inverse(xs))) = ~(R((xr $\cap$ xs)))).*

SPECIAL CLASSES (CONTINUED)

(SP6) *V* is not a set.
(V $\in$ x) $\rightarrow$.
Corollaries.
(V $\in$ (x $\times$ y)) $\rightarrow$.
(<x, V> $\in$ (V $\times$ V)) $\rightarrow$.
(<V, y> $\in$ (V $\times$ V)) $\rightarrow$.

(SP7).
(V = 0) $\rightarrow$.
(V $\subseteq$ 0) $\rightarrow$.

(SP8) Corollaries to (UP2).
% $\rightarrow$ *({x, V} = {x}).*
% $\rightarrow$ *({V, x} = {x}).*

(SP9) Corollary to (SS4).
% $\rightarrow$ *({V} = 0).*

(SP10) Corollaries to (OP3).
$\rightarrow$ *({{x}, {x, 0}} = <x, V>).*
$\rightarrow$ *({0, {{y}}} = <V, y>).*
% $\rightarrow$ *(<V, V> = {0, {0}}).*

(SP11) The class of ordered pairs is not a set.
((V $\times$ V) $\in$ x) $\rightarrow$.

REGULARITY

(RE1) No class can belong to itself.
(x $\in$ x) $\rightarrow$.

(RE2) Corollary to (RE1).
({x} = x) $\rightarrow$.

(RE3) If *memb(x) = x*, then *x* is not a singleton of a set.
({memb(x)} = x), (memb(x) = x) $\rightarrow$.

(RE4) There are no cycles of length 2.
(x $\in$ y), (y $\in$ x) $\rightarrow$.

(RE5) Corollaries to (RE4).
(<x, y> = x) $\rightarrow$.
(<x, y> = y) $\rightarrow$.

(RE6) Converses to (OP10), (OP11) corollaries.
(1st(<x, y>) = <x, y>), (<x, y> $\in$ (V $\times$ V)) $\rightarrow$.
(2nd(<x, y>) = <x, y>), (<x, y> $\in$ (V $\times$ V)) $\rightarrow$.

(RE7) x and its complement can't both be sets.
$(x \in y), (\sim(x) \in z) \rightarrow .$

(RE8) Equivalent conditions for x not to be an ordered pair.
$(1st(x) = x) \rightarrow (2nd(x) = x).$
$(2nd(x) = x) \rightarrow (1st(x) = x).$

(RE9) The components of an ordered pair are sets.
$(<1st(x), 2nd(x)> = x) \rightarrow (1st(x) \in V).$
$(<1st(x), 2nd(x)> = x) \rightarrow (2nd(x) \in V).$
Corollary.
$(<1st(x), 2nd(x)> = x) \rightarrow (x \in (V \times V)).$

(RE10) Corollaries to (RE9).
$(<1st(<x, y>), 2nd(<x, y>)> = <x, y>) \rightarrow (x \in V).$
$(<1st(<x, y>), 2nd(<x, y>)> = <x, y>) \rightarrow (y \in V).$

APPLICATION

(AP1).
$SINGVAL(z), (x \in D(z)) \rightarrow (memb((z \text{“} \{x\})) = (z \text{‘} x)).$
$SINGVAL(z), (x \in D(z)) \rightarrow ((z \text{“} \{x\}) = \{(z \text{‘} x)\}).$

(AP2)-(AP3) Range of z is class of applications of z to domain.

(AP2).
$SINGVAL(z), (x \in D(z)) \rightarrow ((z \text{‘} x) \in R(z)).$

(AP3).
$SINGVAL(z), (y \in R(z)) \rightarrow ((z \text{‘} dom(z, V, y)) = y).$

(AP4).
$(y \in (xf \text{“} \{x\})) \rightarrow (y \subseteq (xf \text{‘} x)).$
Corollaries.
$\rightarrow ((xf \text{“} \{x\}) \subseteq P((xf \text{‘} x))).$
$(<x, y> \in xf), (<x, y> \in (V \times V)) \rightarrow (y \subseteq (xf \text{‘} x)).$

(AP5).
$\rightarrow (((inverse(E) \circ xf) \text{“} \{x\}) = (xf \text{‘} x)).$

(AP6).
$(z \in (xf \text{‘} z)), (z \in diag((inverse(E) \circ xf))) \rightarrow .$
$(z \in V) \rightarrow (z \in (xf \text{‘} z)), (z \in diag((inverse(E) \circ xf))).$

(AP7).
$\rightarrow ((z \text{‘} x) = 0), (x \in D(z)).$

(AP8).
$SINGVAL(xf) \rightarrow ((xf \text{‘} x) \in V).$

(AP9).
$SINGVAL(xf), (<x, y> \in xf), (<x, y> \in (V \times V)) \rightarrow ((xf \text{‘} x) = y).$

(AP10).
$SINGVAL(xf), (x \in D(xf)) \rightarrow (<x, (xf \text{‘} x)> \in xf).$

(AP11).
$SINGVAL(xf), (x \in D(xf)) \rightarrow (((yf \circ xf) \text{‘} x) = (yf \text{‘} (xf \text{‘} x))).$
Corollary.
$SINGVAL(xf), (x \in D((yf \circ xf))) \rightarrow (((yf \circ xf) \text{‘} x) = (yf \text{‘} (xf \text{‘} x))).$

(AP12).
$\rightarrow ((xf \text{‘} x) = 0), (x \in D(xf)).$

(AP13).
$SINGVAL(xf) \rightarrow (((yf \circ xf) ` x) \subseteq (yf ` (xf ` x)))$.

(AP14) Special cases.
$(x \in V) \rightarrow ((V ` x) = V)$.
$\rightarrow ((0 ` x) = 0), (x \in V)$.
$\% \rightarrow ((x ` V) = 0)$.

CANTOR CLASS

(CA1)-(CA3) Alternate definition of Cantor class.

(CA1).
$\rightarrow (cantor(x) \subseteq D(x))$.

(CA2).
$(z \in cantor(xr)), (z \in (xr ` z)) \rightarrow$.

(CA3).
$(z \in D(xr)) \rightarrow (z \in (xr ` z)), (z \in cantor(xr))$.

(CA4) Cantor's Theorem.
$SINGVAL(xf), (D(xf) \in V), (P(D(xf)) \subseteq R(xf)) \rightarrow$.

COMPATIBLE FUNCTIONS

(CF1)-(CF3) Alternate definition of *COMPATIBLE*

(CF1).
$OPERATION(xf), COMPATIBLE(xh, xf, xg) \rightarrow ((D(xh) \times D(xh)) = D(xf))$.

(CF2).
$OPERATION(xg), COMPATIBLE(xh, xf, xg) \rightarrow ((R(xh) \times R(xh)) \subseteq D(xg))$.

(CF3).
$FUNCTION(xh), ((D(xh) \times D(xh)) = D(xf)), ((R(xh) \times R(xh)) \subseteq D(xg)) \rightarrow COMPATIBLE(xh1, xf, xg)$.

(CF4).
$OPERATION(xf), COMPATIBLE(xh, xf, xg), (<x, y> \in (D(xh) \times D(xh))) \rightarrow ((xf ` <x, y>) \in D(xh))$.

(CF5).
$COMPATIBLE(xh, xf, xg), (<x, y> \in (D(xh) \times D(xh))) \rightarrow (<(xh ` x), (xh ` y)> \in D(xg))$.

(CF6).
$COMPATIBLE(xh1, xf1, xg1), COMPATIBLE(xh2, xg1, xg2) \rightarrow (R(xh1) \subseteq D(h2))$.
Corollaries.
$COMPATIBLE(xh1, xf1, xg1), COMPATIBLE(xh2, xg1, xg2) \rightarrow (D((xh2 \circ xh1)) = D(xh1))$.
$COMPATIBLE(xh1, xf1, xg1), COMPATIBLE(xh2, xg1, xg2)$
$\rightarrow ((D((xh2 \circ xh1)) \times D((xh2 \circ xh1))) = (D(xh1) \times D(xh1)))$.

(CF7).
$COMPATIBLE(xh1, xf1, xg1), COMPATIBLE(xh2, xg1, xg2)$
$\rightarrow COMPATIBLE((xh2 \circ xh1), xf1, xg2)$.

HOMOMORPHISMS

(HO1) The composition of homomorphisms is a homomorphism.
$HOM(xh1, xf1, xg1), HOM(xh2, xg1, xg2) \rightarrow HOM((xh2 \circ xh1), xf1, xg2)$.

References

1. Bailin, S., 'A λ-Unifiability Test for Set Theory', *Journal of Automated Reasoning* **4**(3), 269–286 (September 1988).
2. Bernays, P., and Fraenkel, A., *Axiomatic Set Theory*, Amsterdam: North Holland (1958).
3. Bledsoe, W., 'Splitting and Reduction Heuristics in Automatic Theorem Proving', *Artificial Intelligence* **2**(1), 57–78 (1971).
4. 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* **2**(3), 287–327 (1986).
5. Chang, C., and Lee, R., *Symbolic Logic and Mechanical Theorem Proving*, New York: Academic Press (1973).
6. Gödel, K., *The Consistency of the Axiom of Choice and of the Generalized Continuum Hypothesis with the Axioms of Set Theory*, Princeton: Princeton University Press (1940).
7. Hsiang, J., 'Refutational Theorem Proving using Term-Rewriting Systems', *Artificial Intelligence* **25**(3), 255–300 (1985).
8. Knuth, D., and Bendix, P., 'Simple Word Problems in Universal Algebras', in *Computational Problems in Abstract Algebra*, J. Leech (ed.), New York: Pergamon Press (1970).
9. Loveland, D., *Automated Theorem Proving: A Logical Basis*, Amsterdam: North Holland (1978).
10. Lusk, E., and Overbeek, R., 'Experiments with Resolution-Based Theorem-Proving Algorithms', *Computers and Mathematics with Applications* **8**(3), 141–152 (1982).
11. McAllester, D., *Ontic: A Knowledge Representation System for Mathematics*, Cambridge: MIT Press (1989).
12. McCune, W., *Otter 1 0 User's Guide*, ANL-88/44, Argonne National Laboratory (1989).
13. Mendelson, E., *Introduction to Mathematical Logic*, Third Edition, Monterey: Wadsworth & Brooks/Cole (1987).
14. Moravec, H., *Mind Children· The Future of Robot and Human Intelligence*, Cambridge: Harvard University Press (1988).
15. Robinson, J., 'A Machine Oriented Logic Based on the Resolution Principle', *J. Assoc. Comp. Mach.* **12**, 23–41 (1965).
16. Quaife, A., 'Automated Proofs of Löb's Theorem and Gödel's Two Incompleteness Theorems', *Journal of Automated Reasoning* **4**(2), 219–231 (June 1988).
17. Quaife, A., 'Automated Development of Tarski's Geometry', *Journal of Automated Reasoning* **5**(1), 97–118 (March 1989).
18. Quine, W., 'The Problem of Simplifying Truth Functions', *American Mathematical Monthly* **59**(8), 521–531 (October 1952).
19. Suppes, P., *Axiomatic Set Theory*, New York: Dover (1972).
20. Winker, S., and Wos, L., 'Procedure Implementation through Demodulation and Related Tricks', *Proceedings of the Sixth Conference on Automated Deduction* (ed. D. Loveland), New York: Springer-Verlag (1982).
21. Wos, L., Overbeek, R., Lusk, E., and Boyle, J., *Automated Reasoning*, Englewood Cliffs: Prentice Hall (1984).
22. Wos, L., *Automated Reasoning: 33 Basic Research Problems*, Englewood Cliffs: Prentice Hall (1988).