Automated Reasoning: Computer Assisted Proofs in Set Theory Using Gödel's Algorithm for Class Formation

A Thesis
Presented to
The Academic Faculty

by

Tiffany D. Goble

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Mathematics

Georgia Institute of Technology

December 2004

Automated Reasoning: Computer Assisted Proofs in Set Theory
Using Gödel's Algorithm for Class Formation

Approved by:

Dr. Johan G. F. Belinfante, Advisor

Dr. William Green

Dr. Panagiotis Manolios

August 17, 2004

TABLE OF CONTENTS

CHAPTER I

BACKGROUND ON AUTOMATED REASONING IN SET THEORY

1.1 Axiomatization of Set Theory

Early attempts to formulate set theory without the use of an axiom system all led to paradoxes. It has become necessary to adopt a consistent set of axioms in order to develop the ideas of set theory. In the axiomatization of set theory, no definition is used for the concept of a set or the relation between a set and its elements. Instead, a few axioms are assumed and from them everything else must be proven to be true. The first axiomatization of set theory was given by Zermelo[19] in 1908. In this theory the following five axioms are standard: equality axiom, pairing axiom, sum-set (or union) axiom, power-set axiom, and subset axiom. Skolem[11] and Fraenkel[5] independently contributed the axiom of replacement in 1922 as a replacement for the axiom of subsets. This axiom is needed for transfinite induction and ordinal arithmetic.[10]

In contrast to the system described above, von Neumann,[12] Bernays[3] and Gödel[6] developed their own axiom systems of set theory. In Gödel-Bernays class theory, we begin with the premise that everything is a class. Then, we define a class to be a set if there is a class to which it belongs. Classes that are not sets are proper classes. It was postulated by von Neumann that proper classes are those objects that can be mapped onto `V`, the universal class. The Bernays-Gödel system can be proven to be consistent with the Zermelo system above. In the Bernays-Gödel system, the axiom of subsets states that if `x` is a subclass of `y` and `y` is a set, then `x` is a set. The axiom of replacement states that if `f` is a function and `x` is a set, then the image of `x` under `f` is a set.

An additional axiom that can be assumed is the axiom of choice, also called the multiplicative axiom. In this axiom, a set is not uniquely determined by its data, and thus this is not labelled as a constructive axiom. That is, one cannot construct sets from this axiom alone. A couple of questions that were asked early on about the axiom of choice were whether it is consistent with the other axioms and whether it is independent of the other axioms. The answer to the first question was provided by Kurt Gödel,[6] who proved that the axiom of choice is consistent, while the latter question was answered by Paul Cohen,[4] who proved the independence of the axiom of choice. A few other axioms that are involved in this formulation of set theory are the axiom of infinity, the axiom of substitution, and the axiom of foundation.

1.2 Automated Reasoning in Set Theory

Automated reasoning involves the use of computer programs to search for and verify proofs of theorems. The difference between automated reasoning and logical reasoning by humans is that automated reasoning often lacks instantiation and assumptions. It does, however, contain inference rules and make explicit use of strategies.

Some of the most successful applications of automated reasoning are in the field of expert systems. In very specific instances within this field, automated reasoning programs are being used at the same level as human experts. Other applications of automated reasoning programs lie in verifying the performance of computer programs and creating computer programs given certain specifications. They are also applicable to real time systems control, the design and validation of logic circuits, and controlling intelligent robots.

1.2.1 The AURA Program

One of the earliest automated reasoning assistants was named AURA.[18] This was written in IBM 360/370 assembly language and PL / I. This program obtained a lot of the first results in automated reasoning, but it lacked portability because of the language it was written in. In AURA, the user was able to set flags and choose variables for control parameters before executing the program.

1.2.2 The `ITP` Program

Another earlier automated reasoning program was named `ITP`. This program was written in Pascal by Lusk and Overbeek,[15] and provided a user with some interactive use.

1.3 *The* `Otter` *Program*

William McCune of Argonne National Laboratory wrote an automated reasoning program in 1988 named `Otter`. This is written in C and performs somewhat similarly to the AURA program, in that it allows the user to set flags and choose values for control parameters but is not interactive in nature.[16] In fact, both AURA and ITP are predecessors of the Otter program, although `Otter`'s execution time is much faster.

Since `Otter` only draws conclusions if they follow from given hypotheses,[17] there is no chance for logical fallacies or unsound arguments. However, a drawback to `Otter` is that it must assume some conjecture and look for contradictions in order to disprove the negative of the conjecture. This means that `Otter` is used more to prove things the user already suspects is true, rather than being used exploratively to discover new theorems. `Otter` provides many strategies to improve one's chances of success when searching for proofs. One of these strategies is a unit preference strategy, which means that it avoids compound conclusions. Another is a set of support strategy, which helps `Otter` avoid drawing conclusions that are reached only from background hypotheses and thereby helps the program focus on the theorem for which a proof is sought. These strategies, as well as others, help to eliminate many problems that face automated theorem proving programs.

`Otter`'s existence has proven useful in mathematics. In 1996, McCune[9] used the `Otter` program to solve the Robbins Algebra problem, which asks whether the three axioms for a Robbins algebra axiomatize Boolean algebra.

Researchers have provided clausal versions of the von Neumann-Bernays-Gödel set theory that can be programmed into an automated reasoning program. This is possible because the NBG version of set theory has a finite number of axioms, all of which can be input into a computer. After extensive use of the Otter program, Quaife[13] believes that

> "There is no apparent obstacle to the development of set theory through considerably more difficult theorems."

CHAPTER II

THE GOEDEL PROGRAM

2.1 The Mechanics of the GOEDEL Program

The GOEDEL program, written and developed by Dr. Johan G. F. Belinfante,[1] is a computer implementation of Gödel's algorithm for class formation. Mathematica™ is used as an interface for the GOEDEL program. Replacing the axioms for class formation are a few axioms for basic class constructions. All classes must be defined in terms of two basic classes, V and E, and seven basic class constructors: complement, domain, flip, rotate, pairset, cart, and intersection. In the GOEDEL program, V is the universal class and E is the membership relation.

Although the GOEDEL program is not able to carry out deductions automatically, it can verify deductions carried out interactively. Also, in some cases it can prove statements by way of simplifying them to true. Oftentimes it will produce such a result, due to the many simplification rules in the program which are necessary due to the complicated output of Gödel's algorithm. The GOEDEL program contains two quantifiers, forall and exists, but these can only be applied to sets, since Gödel's algorithm does not apply to statements containing quantifiers over classes.

The GOEDEL program serves two important purposes, to reformulate statements containing quantifiers into statements without quantifiers and to act as an interactive reasoning program through with a user can discover new theorems.[2] Once statements are reformulated or discovered, they can then be input into the Otter program in order to obtain clean proofs. The GOEDEL program can also be used to rewrite statements in set theory as equations without variables. Often, this is done with a statement involving class. When class occurs in a statement, Gödel's algorithm is invoked and the simplification rules in the `GOEDEL` program are applied.

Work done in the `GOEDEL` program is done within the framework of the Gödel-Bernays class theory. Because of this, the collection of sets we consider do not have to be sets themselves, but can instead be proper classes.

2.2 Simplification Tools

2.2.1 **`AssertTest`**

`AssertTest` compares a statement to the result after applying `assert` to the statement. The input `assert[p]` will apply Gödel's algorithm to `class[w,p]`, or the class of all `w` such that `p` is true. In this case, the variable `w` must not appear anywhere in `p`. Using `assert` converts a statement into an equation of the form `equal[V,x]` or, equivalently, `equal[0,complement[x]]`. Also, `assert` will convert negative statements into positive ones and will convert statements containing quantifiers into logically equivalent statements without quantifiers.

2.2.2 **`Normality`**

`Normality` is another tool often used to simplify expressions; `Normality[x]` compares `x` with `class[y,member[y,x]]`. Other tools similar to `Normality` are `Renormality`, which is `Normality` with an extra `assert`, and `Rerenormality`, which is `Normality` with two extra `assert`s. There are also similar tests for relations (`RelnNormality`, `RelnRenormality`, and `RelnRerenormality`) and vertical sections (`VSNormality`, `VSRenormality`, and `VSRerenormality`).

2.2.3 **`NotNotTest`**

When `NotNotTest` is applied to a statement in the `GOEDEL` program, it causes the statement to be negated twice, in the course of which the `GOEDEL` program searches for a simplification of the original statement. If no simplification exists, the program simply responds with `True`. Applying `NotNotTest` is particularly successful with statements that contain compound conclusions.

2.2.4 `SubstTest`

`SubstTest` is a tool which allows the user to instantiate a general expression in the `GOEDEL` program and specialize it by making substitutions for the variables. This expression can either be a class or an entire statement. `SubstTest` compares the results of simplifying the expression before and after substitution of the variables and returns `True` if there is no difference between the two resulting expressions. If the two resulting expressions are not exactly the same, then `SubstTest` will return a rule that states that the two expressions are equivalent.

2.2.5 Logical Arguments

Oftentimes, there are too many steps in a proof for the `GOEDEL` program to be able to simplify a statement to true. In these cases, it is necessary to specify a logical argument needed to deduce some statement from others that are known to be true. This is done mainly by using a `SubstTest` and specifying which hypotheses imply which conclusions. If each implication holds under the appropriate substitutions and the logic is sound, the theorem will then be deduced.

CHAPTER III

THE REGULAR CLASS

3.1 *Characterization of* REGULAR

The idea of the axiom of regularity is to prohibit infinite regress of membership statements. In the GOEDEL program, the axiom of regularity is not assumed, but there is a definition for a class of regular sets for which the axiom of regularity holds. The class of regular sets is characterized as follows:

REGULAR = complement[U[DESCENDING]].

The class DESCENDING is characterized as:

DESCENDING = complement[fix[composite[E,DISJOINT]]],

where E is the membership relation and DISJOINT is the class of all pair[x,y] such that x and y are disjoint. U[x] is defined as the union of all sets that belong to x, and fix[x] is the class of all y such that pair[y,y] belongs to x.

Although the GOEDEL program already contained some rewrite rules concerning REGULAR and DESCENDING, there were several more rules available in Otter that needed to be derived within the GOEDEL environment. An overview of the derivation of several of these rules follows.

3.2 *General Theorems About* REGULAR

The universal class is denoted V. To derive that

$$\text{DESCENDING} \in \text{V} \Rightarrow \text{REGULAR} = \text{V},$$

we use a substitution rule that deals with `member[U[U[x]],V]`. This membership rule reduces to `member[x,V]`, or the statement that `x` is a set. This rule is essentially the axiom of sum class. However, if we substitute `DESCENDING` for `x`, a particular rewrite rule will reduce `member[U[U[DESCENDING]],V]` to `equal[REGULAR,V]`. In this way, we derive the theorem.

An important membership theorem is

$$\mathtt{y} \in \mathtt{REGULAR} \;\&\; \mathtt{x} \in \mathtt{y} \;\Rightarrow \mathtt{x} \in \mathtt{REGULAR}.$$

Since membership is not automatically transitive, this takes a little reasoning. However, it is known that

$$\mathtt{y} \in \mathtt{REGULAR} \;\Rightarrow \mathtt{y} \subset \mathtt{REGULAR}.$$

Also, the `GOEDEL` program knows that

$$\mathtt{x} \in \mathtt{y} \;\&\; \mathtt{y} \subset \mathtt{z} \;\Rightarrow \mathtt{x} \in \mathtt{z}.$$

Replacing `z` by `REGULAR`, we can easily see that `x` is a member of `REGULAR`.

It is also relatively easy to show that

$$\mathtt{x} \in \mathtt{DESCENDING} \Rightarrow \mathtt{disjoint[x,REGULAR]}.$$

To derive this, we note

$$\mathtt{x} \in \mathtt{y} \;\Rightarrow \mathtt{x} \subset \mathtt{U[y]}.$$

Replacing `y` by `REGULAR`, we see that

$$\mathtt{x} \in \mathtt{REGULAR} \Rightarrow \mathtt{x} \subset \mathtt{U[DESCENDING]}.$$

The `GOEDEL` program then replaces `U[DESCENDING]` by `complement[REGULAR]`, and from this is able to deduce that `x` and `REGULAR` are disjoint.

3.3 Specific Membership Theorems About REGULAR

Besides general theorems, there are also more specific membership theorems. These take the form that if `x` is a member of `REGULAR`, then `f[x]` is a member of `REGULAR`, where f is some operation acting on x. For instance, using the GOEDEL program we are able to show that

$$\mathtt{x} \in \mathtt{REGULAR} \Rightarrow \mathtt{range[x]} \in \mathtt{REGULAR}.$$

The GOEDEL program will automatically reduce the statement

$$\mathtt{and[member[x, V], subclass[x, REGULAR]]}$$

to member[x,REGULAR]. So we just need to show that if x is a member of REGULAR, then range[x] is a set and a subclass of REGULAR. The fact that range[x] is a set is recognized to be true from the statement that x is a member of REGULAR. We then employ the fact

$$\mathtt{x} \in \mathtt{REGULAR} \Rightarrow \mathtt{x} \subset \mathtt{REGULAR},$$

along with the rule that:

$$\mathtt{x} \subset \mathtt{v} \Rightarrow \mathtt{range[x]} \subset \mathtt{range[v]}.$$

When we replace v by REGULAR, the GOEDEL program will reduce range[REGULAR] to REGULAR and we will have our theorem.

Another specific membership theorem involves deriving that

$$\mathtt{x} \in \mathtt{REGULAR} \Rightarrow \mathtt{rotate[x]} \in \mathtt{REGULAR}.$$

To derive this, we use the rule in the GOEDEL program that states that if x is a subclass of u then rotate[x] is a subclass of rotate[u]. Substituting in REGULAR for u, we get that

$$\mathtt{rotate[x]} \subset \mathtt{cart[cart[REGULAR,REGULAR],REGULAR]}.$$

Since it is already known that cart[REGULAR,REGULAR] is a subclass of REGULAR, it follows that

$$\mathtt{cart[cart[REGULAR,REGULAR],REGULAR]} \subset \mathtt{cart[REGULAR,REGULAR]} \subset \mathtt{REGULAR}.$$

Therefore, we know that rotate[x] is a subclass of REGULAR. Since

$$\mathtt{x} \in \mathtt{V} \Rightarrow \mathtt{rotate[x]} \in \mathtt{V},$$

and

$$\mathtt{v} \in \mathtt{V}\ \&\ \mathtt{v} \subset \mathtt{REGULAR} \Rightarrow \mathtt{v} \in \mathtt{REGULAR},$$

substituting rotate[x] for v we have that

$$\mathtt{x} \in \mathtt{REGULAR} \Rightarrow \mathtt{rotate[x]} \in \mathtt{REGULAR}.$$

The derivation that if x is a member of REGULAR, then composite[x,SWAP] is a member of REGULAR is very similar to the above derivation. We know that if x is a member of REGULAR, then x is a subclass of REGULAR. From the theorems already derived in this section, the GOEDEL program can deduce that if x is a subclass of REGULAR, then composite[x,SWAP] is a subclass of composite[REGULAR,SWAP]. Since composite[REGULAR,SWAP] is known to be a subclass of REGULAR by the GOEDEL program, we can use the rule that

$$\mathtt{u} \subset \mathtt{v}\ \&\ \mathtt{v} \subset \mathtt{w} \Rightarrow \mathtt{u} \subset \mathtt{w}$$

with the replacements

$$\mathtt{u} \to \mathtt{composite[x, SWAP]},$$

$$\mathtt{v} \to \mathtt{composite[REGULAR, SWAP]},$$

and

$$\mathtt{w} \to \mathtt{REGULAR}.$$

Now, we have shown that if x is a member of REGULAR, then composite[x,SWAP] is a subclass of REGULAR. The GOEDEL program does not automatically recognize that if x is a set then composite[x,SWAP] is a set, but this can be shown to be true by the axiom of replacement. This fact combined with the fact that composite[x,SWAP] is contained in REGULAR give us the result that

$$\mathtt{x} \in \mathtt{REGULAR} \Rightarrow \mathtt{composite[x,SWAP]} \in \mathtt{REGULAR}.$$

Along the same lines, we can use the `GOEDEL` program to show that if `x` is a member of `REGULAR`, then `inverse[x]` is also a member of `REGULAR`. We first use the rule that if `x` is a subclass of `y` then `inverse[x]` is a subclass of `inverse[y]`. Substituting `REGULAR` in for y, we obtain the rule that

$$\mathtt{x} \subset \mathtt{REGULAR} \Rightarrow \mathtt{inverse[x]} \subset \mathtt{inverse[REGULAR]}.$$

Since `inverse[REGULAR]` is a subclass of `REGULAR`, this implies that `inverse[x]` is a subclass of `REGULAR` by transitivity of subclass. The `GOEDEL` program already knows that

$$\mathtt{x} \in \mathsf{V} \Rightarrow \mathtt{inverse[x]} \in \mathsf{V}.$$

We again use the rule that

$$\mathtt{v} \in \mathsf{V} \;\&\; \mathtt{v} \subset \mathtt{REGULAR} \Rightarrow \mathtt{v} \in \mathtt{REGULAR},$$

replacing `v` by `inverse[x]` to obtain our final result:

$$\mathtt{x} \in \mathtt{REGULAR} \Rightarrow \mathtt{inverse[x]} \in \mathtt{REGULAR}.$$

One final theorem about members of `REGULAR` that is a bit different from the above is the following:

$$\mathtt{x} \in \mathtt{REGULAR} \;\&\; \mathtt{y} \in \mathtt{REGULAR} \Rightarrow \mathtt{cart[x,y]} \in \mathtt{REGULAR}.$$

Since `x` and `y` are known to be sets from the fact that they are members of `REGULAR`, we need only show that `cart[x,y]` is a subclass of `REGULAR`. We do this by using the fact that

$$\mathtt{u} \subset \mathtt{v} \;\&\; \mathtt{v} \subset \mathtt{w} \Rightarrow \mathtt{u} \subset \mathtt{w}$$

and make the following substitutions:

$$\mathtt{u} \rightarrow \mathtt{cart}[\mathtt{x}, \mathtt{y}],$$

$$\mathtt{v} \rightarrow \mathtt{cart}[\mathtt{REGULAR}, \mathtt{REGULAR}],$$

and

$$\mathtt{w} \rightarrow \mathtt{REGULAR}.$$

From this we obtain the fact that

$$\mathtt{x} \subset \mathtt{REGULAR} \;\&\; \mathtt{y} \subset \mathtt{REGULAR} \;\Rightarrow \mathtt{cart[x,y]} \subset \mathtt{REGULAR},$$

which, with a bit of logical reasoning, completes our derivation.

CHAPTER IV

THE RUSSELL CLASS

4.1 Russell's Paradox

In 1901, Bertrand Russell[14] discovered the Russell paradox. In 1902, this paradox was brought to the attention of Gottlob Frege via a letter[7] from Russell, forcing Frege to reconsider publishing his findings.

The Russell class is the class of all x that do not belong to themselves. In Zermelo-Fraenkel set theory, the axiom scheme states that

$$\forall\, x,\ x \in R \Leftrightarrow x \notin x,$$

where R denotes the Russell class. Since x is a variable, we can replace it with R , to obtain

$$R \in R \Leftrightarrow R \notin R,$$

an obvious contradiction. In the von Neumann and Bernays class theories, this contradiction does not occur. Kelley[8] explains that this is because the Neumann-Bernays axiom scheme states that

$$\forall\, x,\ x \in R \Leftrightarrow x \notin x \,\&\, x \in V.$$

Now, when we replace x by R, we have

$$R \in R \Leftrightarrow R \notin R \,\&\, R \in V,$$

from which we can deduce that the Russell class is not a set.

4.2 Characterization of RUSSELL

The RUSSELL class is the class of all x that do not belong to themselves. The power class P[RUSSELL] of the RUSSELL class has the remarkable property of being a subclass of

RUSSELL,

$$\mathtt{P[RUSSELL]} \subset \mathtt{RUSSELL}.$$

Most of the theorems concerning the RUSSELL class are fairly simple to derive with the GOEDEL program.

4.3 *Theorems About* RUSSELL

It only takes an AssertTest to derive that

$$\mathtt{P[RUSSELL]} \notin \mathtt{x}.$$

By using the rule that

$$\mathtt{u} \in \mathtt{v}\ \&\ \mathtt{v} \subset \mathtt{w}\ \Rightarrow \mathtt{u} \in \mathtt{w}$$

and making the replacements:

$$\mathtt{u} \rightarrow \mathtt{x},$$

$$\mathtt{v} \rightarrow \mathtt{P[RUSSELL]},$$

and

$$\mathtt{w} \rightarrow \mathtt{RUSSELL},$$

we obtain the following theorem,

$$\mathtt{x} \in \mathtt{V}\ \&\ \mathtt{x} \subset \mathtt{RUSSELL}\ \Rightarrow \mathtt{x} \in \mathtt{RUSSELL}.$$

Once we've derived the above theorem, the following can also be derived with only an AssertTest:

$$\mathtt{x} \subset \mathtt{RUSSELL}\ \&\ \mathtt{x} \in \mathtt{V} \Rightarrow \mathtt{succ[x]} \subset \mathtt{RUSSELL},$$

where succ[x] is the union of x and singleton[x].

We can derive that

$$\mathtt{intersection[RUSSELL,x,A[x]]} = 0$$

by using `Renormality`. The class `A[x]` is the class obtained by intersecting all sets that belong to `x`.

To derive a slightly more difficult theorem, we can use the rule that

$$\mathtt{u} \in \mathtt{v} \;\&\; \mathtt{v} \subset \mathtt{w} \;\Rightarrow \mathtt{u} \in \mathtt{w}$$

with the replacements:

$$\mathtt{u} \rightarrow \mathtt{intersection}[\mathtt{RUSSELL}, \mathtt{x}],$$

$$\mathtt{v} \rightarrow \mathtt{P}[\mathtt{RUSSELL}],$$

and

$$\mathtt{w} \rightarrow \mathtt{RUSSELL},$$

which leads to the lemma that

$$\mathtt{intersection[RUSSELL,x]} \in \mathtt{V} \Rightarrow \mathtt{intersection[RUSSELL,x]} \in \mathtt{RUSSELL}.$$

Then, we use the rule that

$$\mathtt{u} \in \mathtt{RUSSELL} \;\Rightarrow \mathtt{u} \notin \mathtt{u},$$

letting `u` be replaced by `intersection[RUSSELL,x]`. From this we obtain the lemma

$$\mathtt{intersection[RUSSELL,x]} \in \mathtt{RUSSELL} \;\Rightarrow \mathtt{intersection[RUSSELL,x]} \notin \mathtt{x}.$$

From the above lemmas we obtain the following theorem:

$$\mathtt{intersection[RUSSELL,x]} \notin \mathtt{x}.$$

The remainder of the theorems about the `RUSSELL` class can also be derived in one step, although the `GOEDEL` program needs a little help with them.

We use the rule

$$\mathtt{u} \subset \mathtt{v} \;\&\; \mathtt{v} \subset \mathtt{w} \;\Rightarrow \mathtt{u} \subset \mathtt{w}$$

with the substitutions

$$\mathtt{u} \rightarrow \mathtt{x},$$

$$\mathtt{v} \rightarrow \mathtt{P}[\mathtt{RUSSELL}],$$

and

$$\mathtt{w} \rightarrow \mathtt{RUSSELL}$$

to obtain the theorem

$$\texttt{U[x]} \subset \mathtt{RUSSELL} \Rightarrow \mathtt{x} \subset \mathtt{RUSSELL}.$$

Using this same rule, but with the replacements

$$\mathtt{u} \to \mathtt{P[x]},$$

$$\mathtt{v} \to \mathtt{P[RUSSELL]},$$

and

$$\mathtt{w} \to \mathtt{RUSSELL},$$

we obtain the theorem

$$\mathtt{x} \subset \mathtt{RUSSELL} \ \Rightarrow \texttt{P[x]} \subset \mathtt{RUSSELL}.$$

To derive the theorem

$$\texttt{P[x]} \subset \texttt{union[RUSSELL,x]},$$

we start with the rule in the `GOEDEL` program that reduces

$$\mathtt{u} \subset \texttt{union[RUSSELL,U[u]]}$$

to `True`. Then we replace `u` by `P[x]`. Since `U[P[x]]` = `x` we obtain our theorem.

The `GOEDEL` program can then use the preceding theorem to derive that

$$\mathtt{RUSSELL} \subset \mathtt{x} \ \Rightarrow \texttt{P[x]} \subset \mathtt{x}.$$

We do this by using the rule in the `GOEDEL` program that says that

$$\mathtt{u} \in \mathtt{v} \ \& \ \mathtt{v} = \mathtt{w} \ \Rightarrow \mathtt{u} \in \mathtt{w}$$

with the replacements

$$\mathtt{u} \to \mathtt{P[x]},$$

$$\mathtt{v} \to \mathtt{union[RUSSELL,x]},$$

and

$$\mathtt{w} \to \mathtt{x}.$$

With these replacements, our theorem is recognized to be true.

CHAPTER V

EQUIVALENCE RELATIONS

5.1 *Characterization of* EQUIVALENCE[x]

EQUIVALENCE[x] is the statement that x is an equivalence relation. In the GOEDEL program, EQUIVALENCE[x] is equivalent to the statement that x is equal to the composite of x and inverse[x] or, alternatively, that x is equal to the composite of inverse[x] and x. It immediately follows from either of these statements that

$$\text{EQUIVALENCE[x]} \Rightarrow \text{x} = \text{inverse[x]}.$$

5.2 *Theorems About* EQUIVALENCE[x]

An interesting rule that the GOEDEL program is already familiar with is the statement that

$$\text{EQUIVALENCE[x]} \;\&\; \text{EQUIVALENCE[u]} \Rightarrow \text{EQUIVALENCE[intersection[x,u]]}.$$

When we specialize this rule by replacing u by cart[y,y], we obtain the following theorem:

$$\text{EQUIVALENCE[x]} \Rightarrow \text{EQUIVALENCE[composite[id[y],x,id[y]]]}.$$

The object id[y] is the restriction of the identity relation to y. So this theorem states that if x is an equivalence relation then the particular restriction of x given above is also an equivalence relation.

If x is an equivalence relation, then we know that x = inverse[x]. We can use this fact with the rule

$$\text{u} \in \text{x} \;\&\; \text{x} = \text{inverse[x]} \Rightarrow \text{u} \in \text{inverse[x]},$$

and specialize u to pair[y,z]. This gives us the theorem that

$$\texttt{EQUIVALENCE[x]} \ \& \ \texttt{pair[z,y]} \in \texttt{x} \Rightarrow \texttt{pair[y,z]} \in \texttt{x}.$$

Since the statement EQUIVALENCE[x] is equivalent to the statement that x is equal to composite[inverse[x],x], we can use the previous theorem to derive

$$\texttt{EQUIVALENCE[x]} \ \& \ \texttt{pair[z,y]} \in \texttt{composite[inverse[x],x]} \Rightarrow \texttt{pair[y,z]} \in \texttt{x}.$$

First we use the GOEDEL rule that

$$\texttt{u} \in \texttt{v} \ \& \ \texttt{v} = \texttt{x} \Rightarrow \texttt{u} \in \texttt{x},$$

with the replacements

$$\texttt{u} \rightarrow \texttt{pair[z,y]}$$

and

$$\texttt{v} \rightarrow \texttt{composite[inverse[x],x]}.$$

This tells us that

$$\texttt{EQUIVALENCE[x]} \ \& \ \texttt{pair[z,y]} \in \texttt{composite[inverse[x],x]} \Rightarrow \texttt{pair[z,y]} \in \texttt{x}.$$

Combined with the previous theorem, this leads to our result.

A similar theorem states that

$$\texttt{EQUIVALENCE[x]} \ \& \ \texttt{pair[y,z]} \in \texttt{x} \Rightarrow \texttt{pair[y,y]} \in \texttt{x}.$$

The derivation of this theorem relies on a lemma. Using an AssertTest we learn that

$$\texttt{pair[y,z]} \in \texttt{x} \ \& \ \texttt{pair[y,z]} \in \texttt{cart[V,V]} \Rightarrow$$
$$\texttt{pair[y,y]} \in \texttt{composite[inverse[x],x]}.$$

Then, the GOEDEL program relies on the previous theorem, in the case where z is replaced by y. This is automatic and does not need to be specified, even in the logical deduction. This, together with the former lemma, derives our theorem.

From the two previous theorems, we are easily able to use SubstTest, along with some logical statements, to derive the following theorem:

$$\mathtt{EQUIVALENCE[x]} \;\&\; \mathtt{pair[y,z]} \in \mathtt{x} \;\Rightarrow \mathtt{pair[z,z]} \in \mathtt{x}.$$

To derive the theorem that

$$\mathtt{EQUIVALENCE[x]} \;\&\; \mathtt{pair[y,z]} \in \mathtt{x} \;\Rightarrow$$
$$\mathtt{image[x,singleton[y]]} = \mathtt{image[x,singleton[z]]},$$

we need a lemma that is significant in its own right. This lemma states that

$$\mathtt{TRANSITIVE[x]} \;\&\; \mathtt{pair[u,v]} \in \mathtt{x} \;\Rightarrow$$
$$\mathtt{image[x,singleton[v]]} \subset \mathtt{image[x,singleton[u]]}.$$

To derive this, we first use an AssertTest on the statement

$$\mathtt{member[pair[u,v],composite[inverse[x],complement[x]]]}.$$

The AssertTest tells us that this statement is equivalent to saying that u is a set and image[x,singleton[v]] is a subclass of image[x,singleton[u]]. Once we have added this replacement to the GOEDEL program, we use the rule

$$\mathtt{w} \in \mathtt{x} \;\&\; \mathtt{x} \subset \mathtt{y} \;\Rightarrow \mathtt{w} \in \mathtt{y}$$

with the substitutions

$$\mathtt{w} \to \mathtt{pair[u,v]}$$

and

$$\mathtt{y} \to \mathtt{composite[Id,complement[composite[inverse[x],complement[x]]]]}.$$

From this we obtain our desired lemma. Once we've added the lemma to the GOEDEL program, we use a SubstTest with a little logical reasoning to obtain our theorem.

Adding all of these new theorems to the GOEDEL program makes the next theorem particularly easy to derive. With just a little reasoning and no new lemmas, we obtain the theorem

$$\mathtt{EQUIVALENCE[x]} \;\&\; \mathtt{pair[z,y]} \in \mathtt{composite[inverse[x],x]} \Rightarrow$$
$$\mathtt{image[x,singleton[y]]} = \mathtt{image[x,singleton[z]]}.$$

CHAPTER VI

PARTIAL ORDERINGS

6.1 *Characterization of* PARTIALORDER[x]

In the GOEDEL program, a partial ordering is characterized by:

REFLEXIVE[x] & TRANSITIVE[intersection[Di,x]].

The above two conditions will automatically reduce to PARTIALORDER[x]. Here Di is the diversity relation and is defined as the class of all pair[x,y] such that x is not equal to y. Reflexive relations are characterized by

$\mathtt{x} \subset$ cart[fix[x],fix[x]] $\Leftrightarrow$ REFLEXIVE[x],

where fix[x] is the class of all y such that pair[y,y] is a member of x. Transitive relations are characterized by

$\mathtt{x} \subset$ composite[Id,complement[composite[complement[x],inverse[x]]]] $\Leftrightarrow$ TRANSITIVE[x].

6.2 *Theorems About* PARTIALORDER[x]

To derive an initial theorem about partial orderings, we use the fact that the GOEDEL program will reduce the statement that u is equal to id[fix[u]] to the statement that u is a subclass of the identity relation. Here, id[fix[x]] is the identity relation restricted to the fixed point set of x. When we replace u by the intersection of x and inverse[x], we get the following rewrite rule:

id[fix[x]] = intersection[x,inverse[x]] $\Rightarrow$
intersection[x,inverse[x]] $\subset$ Id.

This serves to derive the following theorem about partial orderings:

$$\mathtt{PARTIALORDER[x]} \Rightarrow \mathtt{intersection[x,inverse[x]]} = \mathtt{id[fix[x]]}.$$

To derive that

$$\mathtt{PARTIALORDER[x]} \Rightarrow \mathtt{x} = \mathtt{composite[x,x]},$$

we need an important lemma. To derive this lemma, we use the following rewrite rule in the GOEDEL program:

$$\mathtt{u} \subset \mathtt{v} \;\&\; \mathtt{v} \subset \mathtt{u} \Rightarrow \mathtt{u} = \mathtt{v}.$$

When we make the replacements

$$\mathtt{u} \rightarrow \mathtt{x}$$

and

$$\mathtt{v} \rightarrow \mathtt{composite[x,x]},$$

we learn that

$$\mathtt{TRANSITIVE[composite[Id,x]]} \;\&\; \mathtt{x} \subset \mathtt{composite[x,x]} \Rightarrow \mathtt{x} = \mathtt{composite[x,x]}.$$

The phrasing of this rule is a result of the fact that the GOEDEL program reduces subclass[composite
to TRANSITIVE[composite[Id,x]]. However, since

$$\mathtt{PARTIALORDER[x]} \Rightarrow \mathtt{REFLEXIVE[x]} \Rightarrow \mathtt{subclass[x,composite[x,x]]}$$

and

$$\mathtt{PARTIALORDER[x]} \Rightarrow \mathtt{TRANSITIVE[x]} \Rightarrow \mathtt{TRANSITIVE[composite[Id,x]]},$$

we deduce that for partial orderings x is equal to composite[x,x].

The last theorem about partial orderings is somewhat complicated for a few reasons. Not only does it have several hypotheses, but the GOEDEL program has trouble executing the logical argument needed to derive the theorem. Because of this, it is necessary to break the argument into several smaller lemmas. This is a problem that is often encountered when the derivation of a theorem requires several steps. Sometimes this problem can be fixed by 'bundling the hypotheses' into one object rather than having each hypothesis act as its own object. In this case, this did not help and disassembling the derivation was necessary. The first step needed for this derivation uses the fact that the `GOEDEL` program makes the following replacement:

`REFLEXIVE[u]` & `TRANSITIVE[u]` & `intersection[u,inverse[u]]` ⊂ `Id` ⇒ `PARTIALORDER[x]`.

When we replace `u` by `intersection[x,y]`, we obtain the following lemma:

`REFLEXIVE[intersection[x,y]]` & `TRANSITIVE[intersection[x,y]]` & `intersection[x,y,inverse[x],inverse[y]]` ⊂ `Id` ⇒ `PARTIALORDER[intersection[x,y]]`.

The theorem we are trying to derive is

`PARTIALORDER[x]` & `TRANSITIVE[y]` & `REFLEXIVE[y]` ⇒ `PARTIALORDER[intersection[x,y]]`.

In order to derive this theorem, we will need to show that each of the conditions of the above lemma is satisfied by the hypotheses of the theorem. First, we use the rules that

`PARTIALORDER[x]` ⇒ `REFLEXIVE[x]`

and

`REFLEXIVE[x]`& `REFLEXIVE[y]` ⇒ `REFLEXIVE[intersection[x,y]]`

to derive that

`PARTIALORDER[x]` & `REFLEXIVE[y]` ⇒ `REFLEXIVE[intersection[x,y]]`.

Next, we use the same rules but with `TRANSITIVE` instead of `REFLEXIVE` to derive that

`PARTIALORDER[x]` & `TRANSITIVE[y]` ⇒ `TRANSITIVE[intersection[x,y]]`.

For the last piece of this derivation, we use the fact that

$$\mathtt{PARTIALORDER[x]} \Rightarrow \mathtt{intersection[x,inverse[x]]} \subset \mathtt{Id}.$$

Since `intersection[x,inverse[x]]` is a subclass of `Id`, so is any subclass of `intersection[x,inverse[x]]`. It follows that

$$\mathtt{intersection[x,y,inverse[x],inverse[y]]} \subset \mathtt{Id}.$$

With these three pieces and the above lemma, we have completed the derivation of the theorem.

CHAPTER VII

TOTAL ORDERINGS

7.1 *Characterization of* TOTALORDER[x]

In the GOEDEL program, a total ordering is defined by a wrapped rule. Since

$$\texttt{TRANSITIVE[x]} \,\&\, \texttt{ANTISYMMETRIC[x]} \,\&\, \texttt{union[x,inverse[x]]} = \texttt{cart[fix[x],fix[x]]} \Rightarrow \texttt{TOTALORDER[x]},$$

we just need to derive implication in the other direction to find a suitable characterization. Since TOTALORDER[x] implies each of the hypotheses in the above statement, a simple NotNotTest is suitable for proving implication in the other direction. Then, if we choose, we can add to the GOEDEL program the following rule for TOTALORDER[x]:

$$\texttt{TRANSITIVE[x]} \,\&\, \texttt{ANTISYMMETRIC[x]} \,\&\, \texttt{union[x,inverse[x]]} = \texttt{cart[fix[x],fix[x]]} \Leftrightarrow \texttt{TOTALORDER[x]}.$$

It is also true that

$$\texttt{TOTALORDER[x]} \Rightarrow \texttt{PARTIALORDER[x]},$$

so many of the previous theorems can be reformulated to obtain slightly different (and in most cases less general) theorems.

7.2 *Theorems About* TOTALORDER[x]

An easy theorem to derive about total orderings states that

$$\texttt{TOTALORDER[x]} \Rightarrow \texttt{intersection[x,inverse[x]]} \subset \texttt{Id} \,\&\, \texttt{TRANSITIVE[composite[Id,x]]}.$$

To derive this we just need to derive the part that states that for a total ordering `x`, the composite of the identity relation and `x` is transitive. For this, we use the fact that

$$\texttt{TOTALORDER[x]} \Rightarrow \texttt{TRANSITIVE[x]} \Rightarrow \texttt{TRANSITIVE[composite[Id,x]]}.$$

Once we've added this rule to the `GOEDEL` program, we need only perform a `NotNotTest` on the theorem to deduce that it is true.

The next theorem illustrates a way of classifying something as a total ordering. The theorem is as follows

$$\begin{aligned}&\texttt{union[x,inverse[x]]} = \texttt{cart[fix[x],fix[x]]}\ \&\\ &\texttt{intersection[x,inverse[x]]} \subset \texttt{Id}\ \&\\ &\texttt{TRANSITIVE[composite[Id,x]]} \Rightarrow \texttt{TOTALORDER[x]}.\end{aligned}$$

To derive this, we need to derive a lemma. We use the rule in the `GOEDEL` program that says

$$\texttt{u} = \texttt{v} \Rightarrow \texttt{u} \subset \texttt{v},$$

making the replacements

$$\texttt{u} \rightarrow \texttt{union[x, inverse[x]]}$$

and

$$\texttt{v} \rightarrow \texttt{cart[fix[x], fix[x]]}.$$

This is applied in conjunction with the hypothesis that the intersection of `x` and `inverse[x]` is a subclass of the identity to derive that `x` is antisymmetric, a fact later used in the derivation. This lemma, together with some facts the `GOEDEL` program already recognizes as true, completes the derivation of this theorem.

Another theorem about total orderings that can be derived using only rules already recognized by the `GOEDEL` program is:

$$\begin{aligned}&\texttt{union[x,inverse[x]]} = \texttt{cart[fix[x],fix[x]]}\ \&\\ &\texttt{PARTIALORDER[x]} \Rightarrow \texttt{TOTALORDER[x]}.\end{aligned}$$

To derive that

$$\texttt{pair[y,z]} \in \texttt{cart[fix[x],fix[x]]} \;\&\; \texttt{TOTALORDER[x]} \;\Rightarrow$$
$$\texttt{pair[y,z]} \in \texttt{x} \text{ or } \texttt{pair[z,y]} \in \texttt{x},$$

we need an important lemma. Using the rule in the `GOEDEL` program that says that

$$\texttt{u} \in \texttt{v} \;\&\; \texttt{v} = \texttt{w} \;\Rightarrow \texttt{u} \in \texttt{w},$$

we make the replacements

$$\texttt{u} \to \texttt{pair[y,z]},$$
$$\texttt{v} \to \texttt{cart[fix[x],fix[x]]},$$

and

$$\texttt{w} \to \texttt{union[x,inverse[x]]}.$$

Since the `GOEDEL` program already knows that

$$\texttt{TOTALORDER[x]} \;\Rightarrow \texttt{cart[fix[x],fix[x]]} = \texttt{union[x,inverse[x]]},$$

we only need to use a few logical statements to complete this derivation.

One final theorem concerning total orderings deals with the fact that if `x` is a total ordering, then certain restrictions of `x` are also total orderings. More specifically, it says that

$$\texttt{TOTALORDER[x]} \;\Rightarrow \texttt{TOTALORDER[composite[id[y],x,id[y]]]}.$$

We can abbreviate `composite[id[y],x,id[y]]` as `restrict[x,y,y]`, which is the intersection of `x` and `cart[y,y]`. Deriving this theorem involves two lemmas and a bit of logical reasoning. For the first lemma, we use the rule in the `GOEDEL` program that states that

$$\texttt{u} = \texttt{v} \;\Rightarrow \texttt{intersection[u,cart[y,y]]} = \texttt{intersection[v,cart[y,y]]},$$

with the replacements

$$\texttt{u} \to \texttt{union[x,inverse[x]]}$$

and

$$\texttt{v} \to \texttt{cart[fix[x],fix[x]]}.$$

Since, for total orderings, the union of `x` and `inverse[x]` is equal to the cartesian product of `fix[x]` and `fix[x]`, we can now recognize that the result of the above lemma is true for total orderings. For the second lemma, we use the following rule in the `GOEDEL` program:

`PARTIALORDER[u]` & `union[u,inverse[u]]` = `cart[fix[u],fix[u]]` ⇔ `TOTALORDER[u]`,

letting `u` be replaced by `composite[id[y],x,id[y]]`. With these two lemmas, several facts that the `GOEDEL` program is already aware of, and some logical arguments, the derivation of this theorem is complete.

CHAPTER VIII

CONCLUSIONS

While much progress has been made in developing automated reasoning programs, practical applications of these programs to set theory, and mathematics in general, is still in its infancy. `Otter` is a powerful resource for researchers desiring a reasoning assistant, despite its limited interactivity. The `GOEDEL` program will continue to be developed and improved so that it can be used along with the `Otter` program to discover and produce elegant proofs of theorems. It seems likely that the computer assisted approach to theorem proving will continue to be developed to the point that researchers will soon be able to routinely solve open problems that mathematicians have yet been unable to solve.

APPENDIX A

The `REGULAR` Class

The REGULAR Class

```
In[1]:= << "C : \WINDOWS\Desktop\Research\Thesis\
            goedel57.16a";

        << "C : \WINDOWS\Desktop\Research\Thesis\
            Tools.m"
":Package Title:  goedel57.16a 2004 May 16 at 10:05 p.m.  "

It is now:   2004  Jul  14  at  18  :   0

"Loading Simplification Rules"

"TOOLS.M Revised 2004 June 16 "

weightlimit =  40
```

Characterization of REGULAR

```
In[2]:= complement[fix[composite[e, DISJOINT]]]

Out[2]= DESCENDING

In[3]:= complement[U[DESCENDING]]

Out[3]= REGULAR
```

General Theorems About REGULAR

If DESCENDING is a set, then REGULAR is equal to the universal class.

Theorem:

```
In[4]:= SubstTest[member, U[U[x]], V,
           x → DESCENDING] // Reverse
Out[4]= member[DESCENDING, V] == equal[REGULAR, V]
```

```
In[5]:= member[DESCENDING,V] := equal[REGULAR,V]
```

If x is a member of y and y is a member of REGULAR, then x is a member of REGULAR.

Theorem:

```
In[6]:= Map[not,SubstTest[and,implies[p1,p3],
            implies[and[p2,p3],p4],
            not[implies[and[p1,p2],p4]],
            {p1 → member[y,REGULAR],p2 → member[x,y],
              p3 → subclass[y,REGULAR],
              p4 → member[x,REGULAR]}]]
Out[6]= or[member[x,REGULAR],not[member[x,y]],
            not[member[y,REGULAR]]] == True
```

```
In[7]:= or[member[x_,REGULAR],not[member[x_,y_]],
            not[member[y_,REGULAR]]] := True
```

If x is a member of DESCENDING, then x and REGULAR are disjoint.

Theorem:

```
In[8]:= SubstTest[implies,member[x,y],
          subclass[x,U[y]],y → DESCENDING]
Out[8]= or[equal[0,intersection[REGULAR,x]],
            not[equal[0,
                intersection[x,P[complement[x]]]]],
            not[member[x,V]]] == True
```

```
In[9]:= or[equal[0,intersection[REGULAR,x_]],
            not[equal[0,intersection[x_,
                  P[complement[x_]]]]],
            not[member[x_,V]]] := True
```

Restatement:

```
In[10]:= or[not[member[x, DESCENDING]],
           disjoint[x, REGULAR]]
Out[10]= True
```

Specific Membership Theorems Regarding REGULAR

If x is a member of REGULAR, then range[x] is a member of REGULAR.

Lemma 1:

```
In[11]:= SubstTest[implies, subclass[x, y],
           subclass[range[x], range[y]], y → REGULAR]
Out[11]= or[not[subclass[x, REGULAR]],
             subclass[range[x], REGULAR]] == True

In[12]:= (%/.x → x_)/.Equal → SetDelayed
```

Lemma 2:

```
In[13]:= SubstTest[or, member[range[x], P[u]],
           not[member[range[x], V]],
           not[subclass[range[x], u]], u → REGULAR]
Out[13]= or[member[range[x], REGULAR],
             not[member[range[x], V]],
             not[subclass[range[x], REGULAR]]] == True

In[14]:= (%/.x → x_)/.Equal → SetDelayed
```

Theorem:

```
In[15]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p1, p3], implies[p3, p4],
           implies[and[p2, p4], p5],
           not[implies[p1, p5]],
           {p1 → member[x, REGULAR],
             p2 → member[range[x], V],
             p3 → subclass[x, REGULAR],
             p4 → subclass[range[x], REGULAR],
             p5 → member[range[x], REGULAR]}]]
Out[15]= or[member[range[x], REGULAR],
           not[member[x, REGULAR]]] == True
```

```
In[16]:= or[member[range[x_], REGULAR],
           not[member[x_, REGULAR]]] := True
```

If x is a member of REGULAR, then rotate[x] is a member of REGULAR.

Lemma 1:

```
In[17]:= Map[implies[subclass[x, REGULAR], #]&,
          SubstTest[subclass, rotate[x], rotate[u],
           u → REGULAR]]
Out[17]= or[and[subclass[domain[domain[x]], REGULAR],
             subclass[image[x, cart[V, V]], REGULAR],
             subclass[range[domain[x]], REGULAR]],
           not[subclass[x, REGULAR]]] == True
```

```
In[18]:= (%/.x- > x_)/.Equal → SetDelayed
```

Lemma 2a:

```
In[19]:= SubstTest[implies,
           and[subclass[u,v],subclass[v,w]],
           subclass[u,w],
           {u → rotate[REGULAR],
             v → cart[REGULAR,REGULAR],w → REGULAR}]
Out[19]= subclass[cart[cart[REGULAR,REGULAR],
              REGULAR],REGULAR] == True
```

```
In[20]:= subclass[cart[cart[REGULAR,REGULAR],
              REGULAR],REGULAR] := True
```

Lemma 2b:

```
In[21]:= SubstTest[implies,
           and[subclass[u,v],subclass[v,w]],
           subclass[u,w],
           {u → rotate[x],v → rotate[REGULAR],
             w → REGULAR}]
Out[21]= or[
             not[subclass[domain[domain[x]],REGULAR]],
             not[
               subclass[image[x,cart[V,V]],REGULAR]],
             not[subclass[range[domain[x]],REGULAR]],
             subclass[rotate[x],REGULAR]] == True
```

```
In[22]:= (%/.x- > x_)/.Equal → SetDelayed
```

Lemma 3:

```
In[23]:= or[member[rotate[x],REGULAR],
           not[member[rotate[x],V]],
           not[subclass[rotate[x],REGULAR]]]//
         NotNotTest
Out[23]= or[member[rotate[x],REGULAR],
           not[member[rotate[x],V]],
           not[subclass[rotate[x],REGULAR]]] == True

In[24]:= (%/.x->x_)/.Equal→SetDelayed
```

Theorem:

```
In[25]:= Map[not,SubstTest[and,implies[p1,p2],
           implies[p1,p3],implies[p3,p4],
           implies[p4,p5],implies[and[p2,p5],p6],
           not[implies[p1,p6]],
           {p1→member[x,REGULAR],
            p2→member[rotate[x],V],
            p3→subclass[x,REGULAR],
            p4→subclass[rotate[x],rotate[REGULAR]],
            p5→subclass[rotate[x],REGULAR],
            p6→member[rotate[x],REGULAR]}]]
Out[25]= or[member[rotate[x],REGULAR],
           not[member[x,REGULAR]]] == True

In[26]:= or[member[rotate[x_],REGULAR],
           not[member[x_,REGULAR]]] := True
```

If x is a member of REGULAR, then composite[x,SWAP] is a member of REGULAR.

Lemma 1:

```
In[27]:= SubstTest[implies,
           and[subclass[u,v],subclass[v,w]],
           subclass[u,w],
           {u → composite[x,SWAP],
             v → composite[REGULAR,SWAP],w → REGULAR}]
Out[27]= or[
            not[subclass[domain[domain[x]],REGULAR]],
            not[
              subclass[image[x,cart[V,V]],REGULAR]],
            not[subclass[range[domain[x]],REGULAR]],
            subclass[composite[x,SWAP],
              REGULAR]] == True
In[28]:= (%/.x → x_)/.Equal → SetDelayed
```

Lemma 2:

```
In[29]:= Map[
            implies[#,member[composite[x,SWAP],
                  V]]&,member[composite[x,SWAP],V]//
              AssertTest]//Reverse
Out[29]= or[member[composite[x,SWAP],V],
            not[member[domain[domain[x]],V]],
            not[member[image[x,cart[V,V]],V]],
            not[member[range[domain[x]],V]]] == True
In[30]:= (%/.x → x_)/.Equal → SetDelayed
```

```
In[31]:= Map[not,SubstTest[and,implies[p2,p3],
            implies[p3,p4],implies[p3,p5],
            implies[p2,p6],
            implies[and[p4,p5,p6],p7],
            not[implies[p2,p7]],
            {p2 → member[x,V],
              p3 → member[domain[x],V],
              p4 → member[domain[domain[x]],V],
              p5 → member[range[domain[x]],V],
              p6- >member[image[x,cart[V,V]],V],
              p7 → member[composite[x,SWAP],V]}]]
Out[31]= or[member[composite[x,SWAP],V],
            not[member[x,V]]] == True

In[32]:= (%/.x → x_)/.Equal → SetDelayed
```

Lemma 3:

```
In[33]:= SubstTest[implies,and[u,v],w,
            {u → member[composite[x,SWAP],V],
              v → subclass[composite[x,SWAP],REGULAR],
              w → member[composite[x,SWAP],
                  REGULAR]}]//Reverse
Out[33]= or[member[composite[x,SWAP],REGULAR],
            not[member[composite[x,SWAP],V]],
            not[subclass[composite[x,SWAP],
                REGULAR]]] == True

In[34]:= (%/.x → x_)/.Equal → SetDelayed
```

Theorem:

```
In[35]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p2, p3], implies[p3, p4],
           implies[p1, p5], implies[p5, p6],
           implies[and[p4, p6], p7],
           not[implies[p1, p7]],
           {p1 → member[x, REGULAR],
             p2 → subclass[x, REGULAR],
             p3 → subclass[composite[x, SWAP],
                 composite[REGULAR, SWAP]],
             p4 → subclass[composite[x, SWAP],
                 REGULAR], p5 → member[x, V],
             p6 → member[composite[x, SWAP], V],
             p7 → member[composite[x, SWAP],
                 REGULAR]}]]
Out[35]= or[member[composite[x, SWAP], REGULAR],
           not[member[x, REGULAR]]] == True

In[36]:= or[member[composite[x_, SWAP], REGULAR],
           not[member[x_, REGULAR]]] := True
```

If x is a member of REGULAR, then inverse[x] is a member of REGULAR.

Lemma 1:

```
In[37]:= Map[implies[subclass[x, REGULAR], #]&,
          SubstTest[subclass, inverse[x], inverse[v],
           v → REGULAR]]
Out[37]= or[and[subclass[domain[x], REGULAR],
             subclass[range[x], REGULAR]],
           not[subclass[x, REGULAR]]] == True
```

```
In[38]:= (%/.x → x_)/.Equal → SetDelayed
```

Lemma 2:

```
In[39]:= SubstTest[implies,
           and[subclass[u,v],subclass[v,w]],
           subclass[u,w],
           {u → inverse[x],v → inverse[REGULAR],
             w → REGULAR}]
Out[39]= or[not[subclass[domain[x],REGULAR]],
             not[subclass[range[x],REGULAR]],
             subclass[inverse[x],REGULAR]] == True
In[40]:= (%/.x → x_)/.Equal → SetDelayed
```

Lemma 3:

```
In[41]:= Map[
           implies[#,member[inverse[x],REGULAR]]&,
           SubstTest[and,member[u,V],
             subclass[u,REGULAR],u → inverse[x]]]
Out[41]= or[member[inverse[x],REGULAR],
             not[member[domain[x],V]],
             not[member[range[x],V]],
             not[subclass[inverse[x],REGULAR]]] == True
In[42]:= (%/.x → x_)/.Equal → SetDelayed
```

Theorem

```
In[43]:= Map[not, SubstTest[and, implies[p1, p2],
            implies[p2, p3], implies[p3, p4],
            implies[p1, p5], implies[and[p4, p5], p6],
            not[implies[p1, p6]],
            {p1 → member[x, REGULAR],
              p2 → subclass[x, REGULAR],
              p3 → subclass[inverse[x],
                  inverse[REGULAR]],
              p4 → subclass[inverse[x], REGULAR],
              p5- > member[inverse[x], V],
              p6 → member[inverse[x], REGULAR]}]]
Out[43]= or[member[inverse[x], REGULAR],
            not[member[x, REGULAR]]] == True

In[44]:= or[member[inverse[x_], REGULAR],
            not[member[x_, REGULAR]]] := True
```

If x and y are members of REGULAR, then cart[x,y] is a member of REGULAR.

Lemma 1:

```
In[45]:= SubstTest[implies,
          and[subclass[u, v], subclass[v, w]],
          subclass[u, w],
          {u → cart[x, y], v → cart[REGULAR, REGULAR],
            w → REGULAR}]
```

```
Out[45]= or[and[not[equal[0,x]],not[equal[0,y]],
             not[subclass[x,REGULAR]]],
           and[not[equal[0,x]],not[equal[0,y]],
             not[subclass[y,REGULAR]]],
           subclass[cart[x,y],REGULAR]] == True

In[46]:= (%/.{x → x_, y → y_})/.Equal → SetDelayed

In[47]:= Map[not, SubstTest[and, implies[p1, p3],
           implies[p2, p4], implies[and[p3, p4], p5],
           implies[p5, p6],
           not[implies[and[p1, p2], p6]],
           {p1 → member[x, REGULAR],
             p2 → member[y, REGULAR],
             p3 → subclass[x, REGULAR],
             p4 → subclass[y, REGULAR],
             p5 → subclass[cart[x, y],
                 cart[REGULAR, REGULAR]],
             p6 → subclass[cart[x, y], REGULAR]}]]
Out[47]= or[not[member[x,REGULAR]],
           not[member[y,REGULAR]],
           subclass[cart[x,y],REGULAR]] == True

In[48]:= (%/.{x → x_, y → y_})/.Equal → SetDelayed
```

Lemma 2:

```
In[49]:= Map[
          implies[#, member[cart[x, y], REGULAR]]&,
          SubstTest[and, subclass[u, REGULAR],
            member[u, V], u → cart[x, y]]]
```

```
Out[49]= or[and[not[equal[0,x]],not[equal[0,y]],
           not[member[x,V]]],and[not[equal[0,x]],
           not[equal[0,y]],not[member[y,V]]],
          member[cart[x,y],REGULAR],
          not[subclass[cart[x,y],REGULAR]]] == True

In[50]:= (%/.{x → x_, y → y_})/.Equal → SetDelayed
```

Theorem

```
In[51]:= Map[not,SubstTest[and,implies[p1,p3],
          implies[p2,p4],implies[and[p1,p2],p6],
          implies[and[p3,p4],p5],
          implies[and[p5,p6],p7],
          not[implies[and[p1,p2],p7]],
          {p1 → member[x,REGULAR],
           p2 → member[y,REGULAR],p3 → member[x,V],
           p4 → member[y,V],
           p5 → member[cart[x,y],V],
           p6 → subclass[cart[x,y],REGULAR],
           p7 → member[cart[x,y],REGULAR]}]]
Out[51]= or[member[cart[x,y],REGULAR],
          not[member[x,REGULAR]],
          not[member[y,REGULAR]]] == True

In[52]:= or[member[cart[x_,y_],REGULAR],
          not[member[x_,REGULAR]],
          not[member[y_,REGULAR]]] := True
```

APPENDIX B

The `RUSSELL` Class

The RUSSELL Class

```
In[53]:= <<"C:\WINDOWS\Desktop\Research\Thesis\
              goedel57.16a";

         <<"C:\WINDOWS\Desktop\Research\Thesis\
              Tools.m"
":Package Title:  goedel57.16a 2004 May 16 at 10:05 p.m.  "

It is now:   2004  Jul  14  at  18  :   48

"Loading Simplification Rules"

"TOOLS.M Revised 2004 June 16 "

weightlimit =  40
```

Theorems about Russell

The power set of RUSSELL is not a member of x.

Theorem:

```
In[54]:= member[P[RUSSELL],x]//AssertTest

Out[54]= member[P[RUSSELL],x] == False

In[55]:= member[P[RUSSELL],x_] := False
```

If x is a set and x is a subclass of RUSSELL, then x is a member of RUSSELL

Theorem:

```
In[56]:= SubstTest[implies,
           and[member[u,v],subclass[v,w]],
           member[u,w],
           {u → x, v → P[RUSSELL], w → RUSSELL}]
Out[56]= or[member[x,RUSSELL],not[member[x,V]],
            not[subclass[x,RUSSELL]]] == True

In[57]:= or[member[x_,RUSSELL],not[member[x_,V]],
            not[subclass[x_,RUSSELL]]] := True
```

If x is a set and x is a subclass of RUSSELL then the successor of x is a subclass of RUSSELL.

Theorem:

```
In[58]:= or[not[member[x,V]],
            not[subclass[x,RUSSELL]],
            subclass[succ[x],RUSSELL]]//AssertTest
Out[58]= or[not[member[x,V]],
            not[subclass[x,RUSSELL]],
            subclass[succ[x],RUSSELL]] == True

In[59]:= or[not[member[x_,V]],
            not[subclass[x_,RUSSELL]],
            subclass[succ[x_],RUSSELL]] := True
```

The intersection of x, RUSSELL, and A[x] is empty.

Theorem:

```
In[60]:= Map[equal[0,#]&,
           intersection[RUSSELL,x,A[x]]//
             Renormality]
Out[60]= equal[0,intersection[RUSSELL,x,A[x]]] ==
           True

In[61]:= equal[0,intersection[RUSSELL,x_,A[x_]]] :=
           True
```

The intersection of RUSSELL and x not a member of x.

Lemma 1:

```
In[62]:= SubstTest[implies,
           and[member[u,v],subclass[v,w]],
           member[u,w],
           {u → intersection[RUSSELL,x],
             v → P[RUSSELL],w → RUSSELL}]
Out[62]= or[member[intersection[RUSSELL,x],
               RUSSELL],not[member[
                 intersection[RUSSELL,x],V]]] == True

In[63]:= (%/.x → x_)/.Equal → SetDelayed
```

Lemma 2:

```
In[64]:= SubstTest[implies,member[u,RUSSELL],
           not[member[u,u]],
           u → intersection[RUSSELL,x]]
```

```
Out[64]= or[not[member[intersection[RUSSELL,x],
            RUSSELL]],not[member[
            intersection[RUSSELL,x],x]]] == True

In[65]:= (%/.x → x_)/.Equal → SetDelayed
```

Theorem:

```
In[66]:= Map[not[#]&,
           Map[not,SubstTest[and,implies[p1,p2],
               implies[p2,p3],not[implies[p1,p3]],
               {p1- >member[intersection[RUSSELL,x],
                   V],
                p2- >member[intersection[RUSSELL,x],
                   RUSSELL],
                p3 → not[member[intersection[RUSSELL,x],
                    x]]}]]]
Out[66]= member[intersection[RUSSELL,x],x] == False

In[67]:= member[intersection[RUSSELL,x_],x_] :=
           False
```

If U[x] is a subclass of RUSSELL, then x is a subclass of RUSSELL.

Theorem:

```
In[68]:= SubstTest[implies,
           and[subclass[u,v],subclass[v,w]],
           subclass[u,w],
           {u → x,v → P[RUSSELL],w → RUSSELL}]
Out[68]= or[not[subclass[U[x],RUSSELL]],
           subclass[x,RUSSELL]] == True
```

```
In[69]:= or[not[subclass[U[x_],RUSSELL]],
            subclass[x_,RUSSELL]] := True
```

If x is a subclass of RUSSELL then P[x] is a subclass of RUSSELL.

Theorem:

```
In[70]:= SubstTest[implies,
           and[subclass[u,v],subclass[v,w]],
           subclass[u,w],
           {u → P[x],v → P[RUSSELL],w → RUSSELL}]
Out[70]= or[not[subclass[x,RUSSELL]],
            subclass[P[x],RUSSELL]] == True
```

```
In[71]:= or[not[subclass[x_,RUSSELL]],
            subclass[P[x_],RUSSELL]] := True
```

The power class of x is a subclass of the union of x and RUSSELL

Theorem:

```
In[72]:= SubstTest[subclass,u,union[RUSSELL,U[u]],
           u → P[x]]
Out[72]= subclass[P[x],union[RUSSELL,x]] == True
```

```
In[73]:= subclass[P[x_],union[RUSSELL,x_]] :=
           True
```

If RUSSELL is a subclass of x then the power class of x is a subclass of x.

Theorem:

```
In[74]:= SubstTest[implies,
           and[subclass[u,v],subclass[v,w]],
           subclass[u,w],
           {u → P[x],v → union[RUSSELL,x],w → x}]
```

```
Out[74]= or[not[subclass[RUSSELL,x]],
              subclass[P[x],x]] == True

In[75]:= or[not[subclass[RUSSELL,x_]],
              subclass[P[x_],x_]] := True
```

APPENDIX C

Equivalence Relations

Equivalence Relations

```
In[76]:= << "C : \WINDOWS\Desktop\Research\Thesis\
              goedel57.16a";

         << "C : \WINDOWS\Desktop\Research\Thesis\
              Tools.m"
":Package Title:  goedel57.16a 2004 May 16 at 10:05 p.m.  "

It is now:   2004  Jul  14  at  19  :   8

"Loading Simplification Rules"

"TOOLS.M Revised 2004 June 16 "

weightlimit =  40
```

Characterization of EQUIVALENCE[x]

```
In[77]:= equal[x,composite[x,inverse[x]]]

Out[77]= EQUIVALENCE[x]

In[78]:= equal[x,composite[inverse[x],x]]

Out[78]= EQUIVALENCE[x]

In[79]:= implies[EQUIVALENCE[x],equal[x,inverse[x]]]

Out[79]= True
```

Theorems about EQUIVALENCE[x]

If x is an equivalence relation, then composite[id[y],x,id[y]] is an equivalence relation.

Theorem:

```
In[80]:= SubstTest[implies,
           and[EQUIVALENCE[x],EQUIVALENCE[u]],
           EQUIVALENCE[intersection[x,u]],
           u → cart[y,y]]
Out[80]= or[EQUIVALENCE[composite[id[y],x,id[y]]],
            not[EQUIVALENCE[x]]] == True

In[81]:= or[EQUIVALENCE[composite[id[y_],x_,
               id[y_]]],not[EQUIVALENCE[x_]]] :=
           True
```

If x is an equivalence relation and pair[z,y] is a member of x, then pair[y,z] is a member of x.

Lemma:

```
In[82]:= SubstTest[implies,
           and[member[u,x],equal[x,inverse[x]]],
           member[u,inverse[x]],u → pair[z,y]]
Out[82]= or[and[member[y,V],
              member[z,V],member[pair[y,z],x]],
            not[equal[x,inverse[x]]],
            not[member[pair[z,y],x]]] == True

In[83]:= (%/.{x → x_,y → y_,z → z_})/.
           Equal → SetDelayed
```

Theorem

```
In[84]:= Map[not, SubstTest[and, implies[p1, p3],
           implies[and[p2, p3], p4], implies[p4, p5],
           not[implies[and[p1, p2], p5]],
           {p1 → EQUIVALENCE[x],
             p2 → member[pair[z, y], x],
             p3 → equal[x, inverse[x]],
             p4- > member[pair[z, y], inverse[x]],
             p5- > member[pair[y, z], x]}]]
Out[84]= or[member[pair[y, z], x],
           not[EQUIVALENCE[x]],
           not[member[pair[z, y], x]]] == True
```

```
In[85]:= or[member[pair[y_, z_], x_],
           not[EQUIVALENCE[x_]],
           not[member[pair[z_, y_], x_]]] := True
```

If x is an equivalence relation and pair[y,z] is a member of composite[inverse[x],x], then pair[y,z] is a member of x.

Lemma 1:

```
In[86]:= SubstTest[implies,
          and[member[u, v], equal[v, x]], member[u, x],
          {u → pair[z, y],
           v → composite[inverse[x], x]}]
Out[86]= or[member[pair[z, y], x],
           not[EQUIVALENCE[x]],
           not[member[pair[z, y],
               composite[inverse[x], x]]]] == True
```

```
In[87]:= (%/.{x → x_, y → y_, z → z_})/.
           Equal → SetDelayed
```

Theorem:

```
In[88]:= Map[not, SubstTest[and,
            implies[and[p1, p2], p3],
            implies[and[p1, p3], p4],
            not[implies[and[p1, p2], p4]],
            {p1 → EQUIVALENCE[x],
              p2 → member[pair[z, y],
                  composite[inverse[x], x]],
              p3 → member[pair[z, y], x],
              p4 → member[pair[y, z], x]}]]
Out[88]= or[member[pair[y, z], x],
            not[EQUIVALENCE[x]],
            not[member[pair[z, y],
                composite[inverse[x], x]]]] == True
```

```
In[89]:= or[member[pair[y_, z_], x_],
            not[EQUIVALENCE[x_]],
            not[member[pair[z_, y_],
                composite[inverse[x_], x_]]]] := True
```

If x is an equivalence relation and pair[y,z] is a member of x, then pair[y,y] is a member of x.

Lemma:

```
In[90]:= implies[and[member[pair[y, z], x],
              member[pair[y, z], cart[V, V]]],
            member[pair[y, y],
              composite[inverse[x], x]]]//AssertTest
```

```
Out[90]= or[member[pair[y,y],
             composite[inverse[x],x]],
           not[member[y,V]],not[member[z,V]],
           not[member[pair[y,z],x]]] == True
```

```
In[91]:= (%/.{x → x_,y → y_,z → z_})/.
          Equal → SetDelayed
```

Theorem:

```
In[92]:= Map[not,SubstTest[and,implies[p1,p3],
           implies[and[p2,p3],p4],
           implies[and[p2,p4],p5],
           implies[and[p1,p5],p6],
           not[implies[and[p1,p2],p6]],
           {p1 → EQUIVALENCE[x],
             p2 → member[pair[y,z],x],
             p3 → subclass[x,cart[V,V]],
             p4- >member[pair[y,z],cart[V,V]],
             p5 → member[pair[y,y],
                 composite[inverse[x],x]],
             p6- >member[pair[y,y],x]}]]
Out[92]= or[member[pair[y,y],x],
           not[EQUIVALENCE[x]],
           not[member[pair[y,z],x]]] == True
```

```
In[93]:= or[member[pair[y_,y_],x_],
           not[EQUIVALENCE[x_]],
           not[member[pair[y_,z_],x_]]] := True
```

If x is an equivalence relation and pair[y,z] is a member of x, then pair[z,z] is a member of x.

Theorem:

```
In[94]:= Map[not,SubstTest[and,
            implies[and[p1,p2],p3],
            implies[and[p1,p3],p4],
            not[implies[and[p1,p2],p4]],
            {p1 → EQUIVALENCE[x],
              p2 → member[pair[y,z],x],
              p3 → member[pair[z,y],x],
              p4 → member[pair[z,z],x]}]]
Out[94]= or[member[pair[z,z],x],
            not[EQUIVALENCE[x]],
            not[member[pair[y,z],x]]] == True
```

```
In[95]:= or[member[pair[z_,z_],x_],
            not[EQUIVALENCE[x_]],
            not[member[pair[y_,z_],x_]]] := True
```

If x is an equivalence relation and pair[y,z] is a member of x, then image[x,singleton[y]] is equal to image[x,singleton[z]].

Lemma

```
In[96]:= member[pair[u,v],
            composite[inverse[x],complement[x]]]//
          AssertTest
```

```
Out[96]= member[pair[u,v],
              composite[inverse[x],complement[x]]] ==
            and[member[u,V],
              not[subclass[image[x,singleton[v]],
                  image[x,singleton[u]]]]]

In[97]:= member[pair[u_,v_],
              composite[inverse[x_],complement[x_]]] :=
            and[member[u,V],
              not[subclass[image[x,singleton[v]],
                  image[x,singleton[u]]]]]

In[98]:= Map[
            or[subclass[image[x,singleton[v]],
                  image[x,singleton[u]]],#]&,
            SubstTest[implies,
              and[member[w,x],subclass[x,y]],
              member[w,y],
              {w → pair[u,v],
                y → composite[Id,
                    complement[composite[inverse[x],
                        complement[x]]]]}]]
Out[98]= or[not[member[pair[u,v],x]],
              not[TRANSITIVE[x]],
              subclass[image[x,singleton[v]],
                image[x,singleton[u]]]] == True

In[99]:= or[not[member[pair[u_,v_],x_]],
              not[TRANSITIVE[x_]],
              subclass[image[x_,singleton[v_]],
                image[x_,singleton[u_]]]] := True
```

Theorem:

```
In[100]:= Map[not,SubstTest[and,implies[p1,p3],
              implies[and[p2,p3],p4],
              implies[and[p1,p2],p5],
              implies[and[p3,p5],p6],
              implies[and[p4,p6],p7],
              not[implies[and[p1,p2],p7]],
              {p1 → EQUIVALENCE[x],
                p2 → member[pair[y,z],x],
                p3 → TRANSITIVE[x],
                p4 → subclass[image[x,singleton[z]],
                    image[x,singleton[y]]],
                p5 → member[pair[z,y],x],
                p6 → subclass[image[x,singleton[y]],
                    image[x,singleton[z]]],
                p7 → equal[image[x,singleton[y]],
                    image[x,singleton[z]]]}]]
Out[100]= or[equal[image[x,singleton[y]],
                image[x,singleton[z]]],
              not[EQUIVALENCE[x]],
              not[member[pair[y,z],x]]] == True

In[101]:= or[equal[image[x_,singleton[y_]],
                image[x_,singleton[z_]]],
              not[EQUIVALENCE[x_]],
              not[member[pair[y_,z_],x_]]] := True
```

If x is an equivalence relation and pair[y,z] is a member of composite[inverse[x],x], then image[x,singleton[y]] is equal to image[x,singleton[z]].

Theorem:

```
In[102]:= Map[not,SubstTest[and,
              implies[and[p1,p2],p3],
              implies[and[p1,p3],p4],
              not[implies[and[p1,p2],p4]],
              {p1 → EQUIVALENCE[x],
               p2- > not[disjoint[image[x,singleton[y]],
                     image[x,singleton[z]]]],
               p3 → member[pair[y,z],x],
               p4- > equal[image[x,singleton[y]],
                    image[x,singleton[z]]]}]]
Out[102]= or[equal[image[x,singleton[y]],
               image[x,singleton[z]]],
              not[EQUIVALENCE[x]],
              not[member[pair[z,y],
                 composite[inverse[x],x]]]] == True

In[103]:= or[equal[image[x_,singleton[y_]],
               image[x_,singleton[z_]]],
              not[EQUIVALENCE[x_]],
              not[member[pair[z_,y_],
                 composite[inverse[x_],x_]]]] := True
```

APPENDIX D

Partial Orderings

Partial Orderings

```
In[104]:= << "C : \WINDOWS\Desktop\Research\Thesis\
              goedel57.16a";

          << "C : \WINDOWS\Desktop\Research\Thesis\
              Tools.m"
":Package Title:  goedel57.16a 2004 May 16 at 10:05 p.m.  "

It is now:   2004  Jul  14  at  19  :   21

"Loading Simplification Rules"

"TOOLS.M Revised 2004 June 16 "

weightlimit =  40
```

Characterization of PARTIALORDER[x]

```
In[105]:= and[REFLEXIVE[x],
            TRANSITIVE[intersection[Di, x]]]
Out[105]= PARTIALORDER[x]
```

```
In[106]:= subclass[x, cart[fix[x], fix[x]]]

Out[106]= REFLEXIVE[x]
```

```
In[107]:= subclass[x,
            composite[Id,
              complement[composite[complement[x],
                inverse[x]]]]]
Out[107]= TRANSITIVE[x]
```

Theorems about PARTIALORDER[x]

If x is a partial ordering, then the intersection of x and inverse x is equal to the identity restricted to the fixed point set of x.

Lemma:

```
In[108]:= SubstTest[equal,u,id[fix[u]],
           u → intersection[x,inverse[x]]]
Out[108]= equal[id[fix[x]],
            intersection[x,inverse[x]]] ==
           subclass[intersection[x,inverse[x]],Id]
```

```
In[109]:= equal[id[fix[x_]],
            intersection[x_,inverse[x_]]] :=
           subclass[intersection[x,inverse[x]],Id]
```

Restatement:

```
In[110]:= or[not[PARTIALORDER[x]],
           equal[intersection[x,inverse[x]],
            id[fix[x]]]]
Out[110]= True
```

If x is a partial ordering, then x is equal to composite[x,x].

Lemma:

```
In[111]:= Map[implies[#,equal[x,composite[x,x]]]&,
           SubstTest[and,subclass[u,v],
            subclass[v,u],
            {u → x,v → composite[x,x]}]]
```

```
Out[111]= or[equal[x,composite[x,x]],
            not[subclass[x,composite[x,x]]],
            not[TRANSITIVE[composite[Id,x]]]] == True

In[112]:= (%/.x → x_)/.Equal → SetDelayed
```

Theorem:

```
In[113]:= Map[not,SubstTest[and,implies[p1,p2],
            implies[p2,p3],implies[p1,p4],
            implies[p4,p5],implies[and[p3,p5],p6],
            not[implies[p1,p6]],
            {p1 → PARTIALORDER[x],p2 → REFLEXIVE[x],
              p3 → subclass[x,composite[x,x]],
              p4 → TRANSITIVE[x],
              p5 → TRANSITIVE[composite[Id,x]],
              p6- > equal[x,composite[x,x]]}]]
Out[113]= or[equal[x,composite[x,x]],
            not[PARTIALORDER[x]]] == True

In[114]:= or[equal[x_,composite[x_,x_]],
            not[PARTIALORDER[x_]]] := True
```

If x is a partial ordering and y is both reflexive and transitive, then the intersection of x and y is a partial ordering.

Lemma:

```
In[115]:= Map[
            implies[#,PARTIALORDER[
                intersection[x,y]]]&,
            SubstTest[and,REFLEXIVE[u],TRANSITIVE[u],
              subclass[intersection[u,inverse[u]],Id],
              u → intersection[x,y]]]
Out[115]= or[not[REFLEXIVE[intersection[x,y]]],
              not[subclass[intersection[x,y,
                  inverse[x],inverse[y]],Id]],
              not[TRANSITIVE[intersection[x,y]]],
              PARTIALORDER[intersection[x,y]]] == True
In[116]:= (%/.{x → x_,y → y_})/.Equal → SetDelayed
```

This is the proof but it fails to terminate, so we have to break it up:

```
In[117]:= Map[not, SubstTest[and, implies[p1, p4],
            implies[and[p2, p4], p5], implies[p1, p6],
            implies[and[p3, p6], p7], implies[p1, p8],
            implies[p8, p9],
            implies[and[p5, p7, p9], p10],
            not[implies[and[p1, p2, p3], p10]],
            {p1 → PARTIALORDER[x], p2 → REFLEXIVE[y],
             p3 → TRANSITIVE[y], p4 → REFLEXIVE[x],
             p5 → REFLEXIVE[intersection[x, y]],
             p6 → TRANSITIVE[x],
             p7 → TRANSITIVE[intersection[x, y]],
             p8 → subclass[intersection[x, inverse[x]],
                 Id],
             p9- > subclass[intersection[x, y,
                   inverse[x], inverse[y]], Id],
             p10- > PARTIALORDER[intersection[x, y]]}]]
Out[117]= $Aborted
```

Pieces of Proof:

```
In[118]:= Map[not, SubstTest[and, implies[p1, p3],
            implies[and[p2, p3], p4],
            not[implies[and[p1, p2], p4]],
            {p1 → PARTIALORDER[x], p2 → REFLEXIVE[y],
             p3 → REFLEXIVE[x],
             p4 → REFLEXIVE[intersection[x, y]]}]]
Out[118]= or[not[PARTIALORDER[x]], not[REFLEXIVE[y]],
            REFLEXIVE[intersection[x, y]]] == True

In[119]:= (%/.{x → x_, y → y_})/.Equal → SetDelayed
```

```
In[120]:= Map[not,SubstTest[and,implies[p1,p3],
            implies[and[p2,p3],p4],
            not[implies[and[p1,p2],p4]],
            {p1 → PARTIALORDER[x],p2 → TRANSITIVE[y],
              p3 → TRANSITIVE[x],
              p4 → TRANSITIVE[intersection[x,y]]}]]
Out[120]= or[not[PARTIALORDER[x]],not[TRANSITIVE[y]],
            TRANSITIVE[intersection[x,y]]] == True

In[121]:= (%/.{x → x_,y → y_})/.Equal → SetDelayed


In[122]:= Map[not,SubstTest[and,implies[p1,p2],
            implies[p2,p3],not[implies[p1,p3]],
            {p1 → PARTIALORDER[x],
              p2 → subclass[intersection[x,inverse[x]],
                  Id],
              p3- > subclass[intersection[x,y,
                    inverse[x],inverse[y]],Id]}]]
Out[122]= or[not[PARTIALORDER[x]],
            subclass[intersection[x,y,
               inverse[x],inverse[y]],Id]] == True

In[123]:= (%/.{x → x_,y → y_})/.Equal → SetDelayed
```

Putting the pieces together to get the theorem:

```
In[124]:= Map[not,SubstTest[and,
            implies[and[p1,p2],p4],
            implies[and[p1,p3],p5],implies[p1,p6],
            implies[and[p4,p5,p6],p7],
            not[implies[and[p1,p2,p3],p7]],
            {p1 → PARTIALORDER[x],p2 → REFLEXIVE[y],
             p3 → TRANSITIVE[y],
             p4 → REFLEXIVE[intersection[x,y]],
             p5 → TRANSITIVE[intersection[x,y]],
             p6- > subclass[intersection[x,y,
                   inverse[x],inverse[y]],Id],
             p7- > PARTIALORDER[intersection[x,y]]}]]
Out[124]= or[not[PARTIALORDER[x]],
            not[REFLEXIVE[y]],not[TRANSITIVE[y]],
            PARTIALORDER[intersection[x,y]]] == True

In[125]:= or[not[PARTIALORDER[x_]],
            not[REFLEXIVE[y_]],not[TRANSITIVE[y_]],
            PARTIALORDER[intersection[x_,y_]]] :=
          True
```

APPENDIX E

Total Orderings

Total Orderings

```
In[126]:= <<"C:\WINDOWS\Desktop\Research\Thesis\
               goedel57.16a";

          <<"C:\WINDOWS\Desktop\Research\Thesis\
               Tools.m"
":Package Title:  goedel57.16a 2004 May 16 at 10:05 p.m.  "

It is now:   2004  Jul  14  at  19  :   25

"Loading Simplification Rules"

"TOOLS.M Revised 2004 June 16 "

weightlimit =  40
```

Characterization of TOTALORDER[x]

```
In[127]:= implies[TOTALORDER[x],
             and[TRANSITIVE[x],ANTISYMMETRIC[x],
               equal[cart[fix[x],fix[x]],
                 union[x,inverse[x]]]]]//NotNotTest
Out[127]= or[and[equal[cart[fix[x],fix[x]],
                 union[x,inverse[x]]],
               subclass[intersection[x,inverse[x]],
                 Id],TRANSITIVE[x]],
             not[TOTALORDER[x]]] == True

In[128]:= (%/.x → x_)/.Equal → SetDelayed

In[129]:= equiv[TOTALORDER[x],
            and[TRANSITIVE[x],ANTISYMMETRIC[x],
              equal[cart[fix[x],fix[x]],
                union[x,inverse[x]]]]]
```

```
Out[129]= True
```

Theorems about TOTALORDER[x]

If x is a total ordering, then composite[Id,x] is transitive and intersection[x,inverse[x]] is a subclass of Id.

Lemma:

```
In[130]:= Map[not, SubstTest[and, implies[p1, p2],
              implies[p2, p3], not[implies[p1, p3]],
              {p1 → TOTALORDER[x], p2 → TRANSITIVE[x],
               p3 → TRANSITIVE[composite[Id, x]]}]]
Out[130]= or[not[TOTALORDER[x]],
              TRANSITIVE[composite[Id, x]]] == True
In[131]:= (%/.x → x_)/.Equal → SetDelayed
```

Theorem:

```
In[132]:= or[and[subclass[intersection[x, inverse[x]],
                Id], TRANSITIVE[composite[Id, x]]],
              not[TOTALORDER[x]]]//NotNotTest
Out[132]= or[and[subclass[intersection[x, inverse[x]],
                Id], TRANSITIVE[composite[Id, x]]],
              not[TOTALORDER[x]]] == True

In[133]:= or[
              and[subclass[intersection[x_,
                   inverse[x_]], Id],
               TRANSITIVE[composite[Id, x_]]],
              not[TOTALORDER[x_]]] := True
```

If cart[fix[x],fix[x]] is equal to union[x,inverse[x]] , intersection[x,inverse[x]] is a subclass of Id, and composite[Id,x] is transitive, then x is a total ordering.

Lemma:

```
In[134]:= SubstTest[implies, equal[u, v],
            subclass[u, v],
            {u → union[x, inverse[x]],
             v → cart[fix[x], fix[x]]}]
Out[134]= or[and[REFLEXIVE[x],
               REFLEXIVE[composite[Id, x]]],
             not[equal[cart[fix[x], fix[x]],
                 union[x, inverse[x]]]]] == True
In[135]:= (%/.x → x_)/.Equal → SetDelayed
```

Theorem:

```
In[136]:= Map[not, SubstTest[and, implies[p3, p4],
              implies[p4, p5], implies[and[p1, p5], p6],
              implies[and[p2, p3, p6], p7],
              not[implies[and[p1, p2, p3], p7]],
              {p1 → subclass[intersection[x, inverse[x]],
                   Id], p2 → TRANSITIVE[composite[Id, x]],
               p3 → equal[union[x, inverse[x]],
                   cart[fix[x], fix[x]]],
               p4 → subclass[union[x, inverse[x]],
                   cart[fix[x], fix[x]]],
               p5 → subclass[x, cart[V, V]],
               p6- > ANTISYMMETRIC[x],
               p7 → TOTALORDER[x]}]]
```

```
Out[136]= or[not[equal[cart[fix[x],fix[x]],
              union[x,inverse[x]]]],not[subclass[
              intersection[x,inverse[x]],Id]],
          not[TRANSITIVE[composite[Id,x]]],
          TOTALORDER[x]] == True
```

```
In[137]:= or[not[equal[cart[fix[x_],fix[x_]],
              union[x_,inverse[x_]]]],
          not[subclass[intersection[x_,
               inverse[x_]],Id]],
          not[TRANSITIVE[composite[Id,x_]]],
          TOTALORDER[x_]] := True
```

If x is a partial ordering and cart[fi x[x],fi x[x]] is equal to union[x,inverse[x]], then x is a total ordering.

Theorem:

```
In[138]:= Map[not,SubstTest[and,implies[p2,p3],
          implies[p2,p4],
          implies[and[p1,p3,p4],p5],
          not[implies[and[p1,p2],p5]],
          {p1-> equal[union[x,inverse[x]],
               cart[fix[x],fix[x]]],
            p2 → PARTIALORDER[x],p3 → TRANSITIVE[x],
            p4 → subclass[intersection[x,inverse[x]],
               Id],p5 → TOTALORDER[x]}]]
Out[138]= or[not[equal[cart[fix[x],fix[x]],
              union[x,inverse[x]]]],
          not[PARTIALORDER[x]],
          TOTALORDER[x]] == True
```

```
In[139]:= or[not[equal[cart[fix[x_],fix[x_]],
                  union[x_,inverse[x_]]]],
            not[PARTIALORDER[x_]],TOTALORDER[x_]] :=
          True
```

If pair[y,z] is a member of cart[fix[x],fix[x]], x is a total ordering, // and pair[z,y] is not a member of x, then pair[y,z] is a member of x.

Lemma:

```
In[140]:= SubstTest[implies,
            and[member[u,v],equal[v,w]],
            member[u,w],
            {u → pair[y,z],v → cart[fix[x],fix[x]],
             w → union[x,inverse[x]]}]

Out[140]= or[and[member[y,V],
              member[z,V],member[pair[z,y],x]],
            member[pair[y,z],x],
            not[equal[cart[fix[x],fix[x]],
                union[x,inverse[x]]]],
            not[member[y,fix[x]]],
            not[member[z,fix[x]]]] == True

In[141]:= (%/.{x → x_,y → y_,z → z_})/.
          Equal- > SetDelayed
```

Theorem:

```
In[142]:= Map[not,SubstTest[and,implies[p3,p4],
            implies[and[p2,p4],p5],
            implies[and[p1,p5],p6],
            not[implies[and[p1,p2,p3],p6]],
            {p1 → not[member[pair[z,y],x]],
              p2- >member[pair[y,z],
                  cart[fix[x],fix[x]]],
              p3 → TOTALORDER[x],
              p4 → equal[cart[fix[x],fix[x]],
                  union[x,inverse[x]]],
              p5- >member[pair[y,z],
                  union[x,inverse[x]]],
              p6- >member[pair[y,z],x]}]]
Out[142]= or[member[pair[y,z],x],
            member[pair[z,y],x],
            not[member[y,fix[x]]],
            not[member[z,fix[x]]],
            not[TOTALORDER[x]]] == True

In[143]:= or[member[pair[y_,z_],x_],
            member[pair[z_,y_],x_],
            not[member[y_,fix[x_]]],
            not[member[z_,fix[x_]]],
            not[TOTALORDER[x_]]] := True
```

If x is a total ordering, then composite[id[y],x,id[y]] is a total ordering.

Lemma 1:

```
In[144]:= SubstTest[implies,equal[u,v],
            equal[intersection[u,cart[y,y]],
              intersection[v,cart[y,y]]],
            {u → union[x,inverse[x]],
              v → cart[fix[x],fix[x]]}]
Out[144]= or[equal[cart[intersection[y,fix[x]],
                intersection[y,fix[x]]],
              union[composite[id[y],x,id[y]],
                composite[id[y],inverse[x],id[y]]]],
            not[equal[cart[fix[x],fix[x]],
                union[x,inverse[x]]]]] == True

In[145]:= (%/.{x → x_,y → y_})/.Equal- > SetDelayed
```

Lemma 2:

```
In[146]:= SubstTest[implies,
            and[PARTIALORDER[u],
              equal[union[u,inverse[u]],
                cart[fix[u],fix[u]]]],TOTALORDER[u],
            u → composite[id[y],x,id[y]]]
Out[146]= or[not[equal[cart[intersection[y,fix[x]],
                  intersection[y,fix[x]]],
                union[composite[id[y],x,id[y]],
                  composite[id[y],inverse[x],id[y]]]]],
              not[PARTIALORDER[composite[id[y],
                  x,id[y]]]],TOTALORDER[
                composite[id[y],x,id[y]]]] == True
```

```
In[147]:= (%/.{x → x_, y → y_})/.Equal-> SetDelayed
```

Theorem:

```
In[148]:= Map[not, SubstTest[and, implies[p1, p2],
            implies[p1, p3], implies[p2, p4],
            implies[p3, p5], implies[and[p4, p5], p6],
            not[implies[p1, p6]],
            {p1 → TOTALORDER[x], p2 → PARTIALORDER[x],
             p3 → equal[union[x, inverse[x]],
                 cart[fix[x], fix[x]]],
             p4 → PARTIALORDER[
                 composite[id[y], x, id[y]]],
             p5-> equal[cart[intersection[y, fix[x]],
                  intersection[y, fix[x]]],
                 union[composite[id[y], x, id[y]],
                  composite[id[y], inverse[x], id[y]]]],
             p6 → TOTALORDER[
                 composite[id[y], x, id[y]]]}]]

Out[148]= or[not[TOTALORDER[x]], TOTALORDER[
             composite[id[y], x, id[y]]]] == True

In[149]:= or[not[TOTALORDER[x_]],
            TOTALORDER[composite[id[y_], x_,
              id[y_]]]] := True
```

REFERENCES

[1] Belinfante, Johan G. F., Gödel's Algorithm for Class Formation, Journal of Symbolic Computation, volume 1831 (1990), pp. 132–147.

[2] Belinfante, Johan G. F. and Goble, Tiffany D., CORE and HULL Constructors in Gödel's Class Theory, IJCAR 2004 Workshop 7 on Computer-Supported Mathematical Theory Development held July 5, 2004 in Cork Ireland, Benzmüller, Christoph and Windsteiger,Wolfgang (Chairs), pp 73–89.

[3] Bernays, Paul, Axiomatic Set Theory with a Historical Introduction by Abraham A. Fraenkel, North-Holland Publishing Company, Amsterdam, 1958.

[4] Cohen, P. J., Set Theory and the Continuum Hypothesis, W. A. Benjamin, New York, 1966.

[5] Fraenkel, A. A., On the Foundations of the Cantor-Zermelo Set Theory, Math. Annal. 86 (1922), pp. 230–237.

[6] Gödel, K., The consistency of the axiom of choice and of the generalized continuum-hypothesis with the axioms of set theory, Princeton University Press, Princeton, 1940.

[7] Heijenoort, Jean von, From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931, Harvard University Press, Cambridge, 1967.

[8] Kelley, John L., General Topology, D. Van Nostrand, Company Inc., Princeton, 1955.

[9] McCune, W., Solution of the Robbins problem, J. Automated Reasoning, 19(3) (1997), pp. 263–276.

[10] Rubin, Jean E., Set Theory for the Mathematician, Holden-Day, San Francisco, 1967.

[11] Skolem, T., Some remarks on the axiomatic foundations of set theory, Wiss. Vorträge gehalten aug dem 5. Kongress der Skandenav. Mathemaliken in Helsingfors (1922), pp. 217–232.

[12] Neumann, John von, The Axiomatisation of Set Theory, Math. Zschr. 27, Vol. I, No. 16(1928), pp. 669–752.

[13] Quaife, Art, Automated Development of Fundamental Mathematical Theories Kluwer Academic Publishers, Dordrecht, 1992.

[14] Russell, B., On some difficulties in the theory of transfinite numbers and order types, Proc. London Math. Soc. (1906), (2), 4, pp. 29–53.

[15] Wos, L., Overbeek, R., Lusk, E. and Boyle, J., Automated Reasoning: Introduction and Applications, 2nd ed., McGraw-Hill, New York, 1992.

[16] Wos, Larry and Pieper, Gail W., The Collected Works of Larry Wos, Volume 1, Exploring the Power of Automated Reasoning, World Scientific, Singapore, 2000.

[17] Wos, Larry and Pieper, Gail W., The Collected Works of Larry Wos, Volume 2, Applying Automated Reasoning to Puzzles, Problems, and Open Questions, World Scientific, Singapore, 2000.

[18] Wos, Larry and Pieper, Gail W., A Fascinating Country in the World of Computing: Your Guide to Automated Reasoning, World Scientific, Singapore, 1999.

[19] Zermelo, E., Investigations on the Foundations of Set Theory I, Math. Annal., 65 (1908), pp. 261–281.