Sets and Computers:
Summary of Summer 2008 REU Project

William Drobny
August 1, 2008

```
SetDirectory["C:\Research"] << goedel.08jul29r.m; << tools.m;
:Package Title: goedel.08jul29r                2008 July 29 at 4:20 p.m.
It is now:  2008 Aug 1 at 14:13
Loading Simplification Rules
TOOLS.M                                    Revised 2008 May 17
weightlimit = 40
```

Dedication

To Steve Sigur, teacher and friend. Your lessons will not be forgotten.

Introduction

The Research Experience for Undergraduates (REU) program is an opportunity for students to work with professors on common research interests. The REU program is funded indirectly by the National Science Foundation (NSF) to instill the importance of research in undergraduate students. This summer, I worked with Professor Belinfante on his research in set theory using automated reasoning. This report gives a brief summary of the field, as well as explaining some of the progress made this summer. I would like to thank Professor Matt Baker for giving me the opportunity to do an REU and Professor Johan Belinfante for mentoring me this summer.

Automated Reasoning

Automated reasoning is the use of computers to help prove theorems. There are two types of automated theorem provers, fully automated and interactive. In both branches of automated reasoning, the theorem prover manipulates axioms and a list of previously proven theorems to prove theorems and amend its theorem bank; however, the way the programs operates vary greatly. Fully automated theorem provers (such as William McCune's programs **Otter** and its successor **Prover9**) search for theorems by methodically evaluating all combinations of previously known facts until a proof is found. This method works because of Gödel's Completeness Theorem, which states that a theorem can be proven from a set of axioms iff it can be logically deduced from mechanical applications of the axioms and the rules of inference. Despite producing many unwanted results en route to finding a proof, they often work at speeds much faster than interactive programs. Because the person operating the theorem prover has less input over what the program does in searching for a proof, fully automated theorem provers sometimes find elegant proofs that mathematicians might not find. One of the downsides of fully automated theorem provers is that unless sufficient care is taken, they may run into a combinatorial explosion of possibilities to try when looking for a proof. Interactive theorem provers allow the user more guidance over the program; however, this control comes at a cost. Interactive theorem provers require one to have a well-formulated strategy in order to prove a theorem.

The **GOEDEL** program is an interactive theorem prover in the form of a Mathematica™ package which allows users to manipulate expressions to produce equivalent expressions. When two formulas for a class are proven to be equal using the tools available, a rewrite rule can be made that simplifies one formula to the other. Similarly, when two statements about classes are proven to be logically equivalent, a rewrite rule can be made that simplifies one of them. One of the ways lemmas and theorems are proven is using the **SubstTest** tool, which compares two orders of evaluation of a statement or two ways of simplifying a formula for a class, therefore preserving equivalence or equality while producing different expressions. **AssertTest** is another useful tool because it compares a statement **p** to the statement that the universal class is equal to the class of all **x** such that **p** when **x** is a variable that does not appear in the statement **p**, again preserving logical equivalence while possibly producing new results. Other tools exist that evaluate expressions in different, yet equivalent orders and comparing the result.

Rewrite Rules

When a theorem is proven in the **GOEDEL** program, often a rewrite rule can be created causing one of two equivalent expressions to be automatically changed to the other when it is encountered. One of the problems with automated reasoning is the fear of an infinite loop that the computer tries to simplify forever. If two similar rewrite rules have reverse direction, or if a rewrite rule is self-referential and expanding (i.e., **x → and[x, y]**), then the program will loop. Here is a concrete example of how a poorly oriented rewrite rule causes an infinite loop.

Lemma.

```
implies[equal[x, set[x]], member[x, x]] // AssertTest
```

```
or[member[x, x], not[equal[x, set[x]]]] == True
```

```
or[member[x_, x_], not[equal[x_, set[x_]]]] := True
```

Theorem.

```
SubstTest[and, p, implies[p, q], {p → equal[x, set[x]], q → member[x, x]}] // Reverse
```

```
equal[x, set[x]] == and[equal[x, set[x]], member[x, x]]
```

Suppose one were to foolishly add this rewrite rule:

```
equal[x_, set[x_]] := and[equal[x, set[x]], member[x, x]]
```

This is what happens when the program encounters an infinite loop.

```
equal[x, set[x]]
```

```
$RecursionLimit::reclim : Recursion depth of 256 exceeded. ≫

$RecursionLimit::reclim : Recursion depth of 256 exceeded. ≫

$RecursionLimit::reclim : Recursion depth of 256 exceeded. ≫

General::stop : Further output of $RecursionLimit::reclim will be suppressed during this calculation. ≫

and[equal[x,
  Hold[RuleCondition[$ConditionHold[$ConditionHold[0]], simplify && not[member[U[x], V]]]]],
 Hold[RuleCondition[$ConditionHold[$ConditionHold[False]], AxReg]], member[x, x]]
```

The bad rewrite rule can be removed:

```
equal[x_, set[x_]] =.
```

When a new result is found, an important decision arises: how should the rewrite rule be oriented? A general rule of thumb is to make expressions become "simpler", and in many cases, such as theorems simplifying to **True**, this is a very easy decision; however, when a rule equates two expressions involving different concepts, it may be difficult to decide which expression is simpler. One way to compare the "simplicity" of expressions is to compare their weights, a measure of complexity based upon the terms and functors used in an expression, but weight is not the only consideration. How a new rule would interact with other rules is important. One fear is that a new rule could cause infinite looping with an old similar rule oriented in the opposite direction. Ideally, one hopes for some synergy with a new rewrite rule, where old rules cause certain expressions rewrite to the left side of the rule or further simplify the right side. Still, sometimes its better to modify a prospective rule using **Map** and save the alteration as a rewrite rule. For example, if two expressions are proven equal, if may be better to rewrite a statement of their equality to **True** rather than to rewrite one expression to the other.

Classes and Class Constructors

In the Gödel-Bernays axiomatization of class theory sets are defined as classes that can belong to other classes. A proper class is a class that is not a set. A proper class cannot be a member of another class. One of the problems all axiomatizations of set theory must answer is how to deal with Russell's Paradox, which proposes if the class containing all non-self-containing classes contains itself.

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

```
RUSSELL
```

Russell's paradox is avoided by the fact that the class **RUSSELL** of all sets that do not belong to themselves is a proper class.

```
member[RUSSELL, V]
```

```
False
```

Another example of a proper class is the class of all sets. This class is called the universal class **V**.

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

```
V
```

There are many constructors that build a new class from another. For example, the power class **P[x]** of any class **x** is the class of all subsets of **x**.

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

```
P[x]
```

Similarly, the sum class **U[x]** of any class **x** is the class that holds all members of the members of **x**.

```
class[y, exists[z, and[member[y, z], member[z, x]]]]
```

```
U[x]
```

One interesting fact is that the sum class of the power class of a class is itself.

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

```
x
```

One property that a class may have is fullness, the condition that all members of the class are also subsets.

```
equiv[full[x], assert[forall[y, implies[member[y, x], subclass[y, x]]]]]
```

```
True
```

Two other class constructors are the hereditarily core **H[x]** and the transitive closure **tc[x]** of a class **x**. The former can be defined by:

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

```
H[x]
```

A direct definition of **tc[x]** is more complex: one could use the transitive closure function **TC** defined by:

```
VERTSECT[complement[composite[inverse[complement[E]], id[FULL], S]]]
```

```
TC
```

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

```
tc[x]
```

One can characterize the transitive closure of **x** as the smallest full class containing **x**.

```
implies[and[subclass[x, z], full[z]], subclass[tc[x], z]]
```

```
True
```

```
not[exists[z, and[propersubclass[z, tc[x]], subclass[x, z], full[z]]]] // assert
```

```
True
```

Similarly, the hereditarily core of class **x** is the largest full subclass of **x**.

```
implies[and[subclass[z, x], full[z]], subclass[z, H[x]]]
```

```
True
```

```
not[exists[z, and[propersubclass[H[x], z], subclass[z, x], full[z]]]] // assert
```

```
True
```

The fixed point class **fix[x]** of a relation **x** is the class of all sets that are self-related by **x**.

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

```
fix[x]
```

The **fix** constructor preserves all Boolean operations: complement, union, and intersection.

```
fix[complement[x]]
```

```
complement[fix[x]]
```

```
fix[union[x, y]]
```

```
union[fix[x], fix[y]]
```

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

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

The Axiom of Regularity

A commonly accepted axiom of set theory is the Axiom of Regularity, which states that there can be no infinitely descending chain of membership. To avoid the concept of "infinite" in the statement of the axiom of regularity, one can reformulate the Axiom of Regularity as a non-recycling condition on membership: let us say that a class **x** of sets is a descending class if for every member **u** of **x**, there is a member **v** of **x** such that **v** belongs to **u**.

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

```
DESCENDING
```

The Axiom of Regularity asserts that the only descending set is the empty set.

```
equal[DESCENDING, set[0]]
```

```
AxReg
```

If the Axiom of Regularity is assumed, the Russell class is equal to the universal class **V**.

```
implies[AxReg, equal[V, RUSSELL]]
```

```
True
```

However, if one denies the Axiom of Regularity, several interesting questions arise. Define the class **REGULAR** to be the complement of the union of all descending sets.

```
complement[U[DESCENDING]]
```

```
REGULAR
```

The Axiom of Regularity is equivalent to the statement that all sets are in **REGULAR**.

```
equal[REGULAR, V]
```

```
AxReg
```

This class has many remarkable properties. For example, it is its own power class and its own sum class :

```
P[REGULAR]
```

```
REGULAR
```

```
U[REGULAR]
```

```
REGULAR
```

Also, **REGULAR** is a full class, so **H[REGULAR] = REGULAR = tc[REGULAR]**.

```
full[REGULAR]
```

```
True
```

```
H[REGULAR]
```

```
REGULAR
```

```
tc[REGULAR]
```

```
REGULAR
```

Irregularity of Self-Membership

When the Axiom of Regularity is denied, the **RUSSELL** and **REGULAR** classes no longer equal **V**, making them topics of study. The question of which classes are contained in which others becomes a matter of interest. One conclusion is that **P[RUSSELL]** is a subclass of **RUSSELL**. To show this, we temporarily remove the rewrite rule that asserts this fact, and reprove it.

```
subclass[P[RUSSELL], RUSSELL] =.
```

The theorem can be proven in three simple steps.

```
SubstTest[equiv, or[p1, p2], or[p1, not[p2]],
 {p1 → implies[member[x, x], not[member[x, P[RUSSELL]]]], p2 → member[x, RUSSELL]}]
```

```
or[not[member[x, x]], not[subclass[x, RUSSELL]]] == True
```

```
Map[class[x, #] &, %]
```

```
union[RUSSELL, complement[P[RUSSELL]]] == V
```

```
Map[equal[V, #] &, %]
```

```
subclass[P[RUSSELL], RUSSELL] == True
```

These three steps can be compounded into a single line:

```
Map[equal[V, class[x, #]] &, SubstTest[equiv, or[p1, p2], or[p1, not[p2]],
  {p1 → implies[member[x, x], not[member[x, P[RUSSELL]]]], p2 → member[x, RUSSELL]}]]
```

```
subclass[P[RUSSELL], RUSSELL] == True
```

Now that the theorem is reproven, one can add the appropriate rewrite rule.

```
subclass[P[RUSSELL], RUSSELL] := True
```

From this, and the fact that the power class constructor preserves the subclass relation, it is apparent that **P[P[RUSSELL]]** is a subclass of **P[RUSSELL]**, and the iteration of the power class constructor produces a descending string of subclasses:

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

```
True
```

```
subclass[P[P[RUSSELL]], P[RUSSELL]]
```

```
True
```

```
subclass[P[P[P[RUSSELL]]], P[P[RUSSELL]]]
```

```
True
```

Although **H[RUSSELL]** is the largest full subclass of **RUSSELL**, it is more transparently characterized as the class of all sets with no self-membership in their transitive closure.

```
class[x, not[exists[y, and[member[y, tc[x]], member[y, y]]]]]
```

```
H[RUSSELL]
```

At the "end" of this string is **H[RUSSELL]**.

H[RUSSELL] ⊂ ... ⊂ P[P[P[RUSSELL]]]⊂ P[P[RUSSELL]]⊂ P[RUSSELL]⊂ RUSSELL

This is because **H[RUSSELL]** is its own power class.

```
P[H[RUSSELL]]
```

```
H[RUSSELL]
```

```
subclass[H[RUSSELL], RUSSELL]
```

```
True
```

```
subclass[H[RUSSELL], P[RUSSELL]]
```

```
True
```

```
subclass[H[RUSSELL], P[P[RUSSELL]]]
```

```
True
```

```
subclass[H[RUSSELL], P[P[P[RUSSELL]]]]
```

```
True
```

Now that we know the structure of the Russell power class string, one would like to know when equality could exist or what counter examples cause the subclasses to be proper subclasses. If f**ix[E]**, the complement of **RUSSELL** is non empty, then the string of **RUSSELL** subclasses becomes a string of proper subclasses. Roughly, the proof follows from the following two statements.

```
implies[not[empty[fix[E]]], propersubclass[P[RUSSELL], RUSSELL]] // assert
```

```
True
```

```
implies[not[empty[fix[E]]], propersubclass[P[P[RUSSELL]], P[RUSSELL]]] // assert
```

```
True
```

```
implies[not[empty[fix[E]]], propersubclass[P[P[P[RUSSELL]]], P[P[RUSSELL]]]] // assert
```

```
True
```

Also, if there is a set that belongs to itself, then **H[RUSSELL]** is a proper subclass of **RUSSELL**.

```
implies[not[empty[fix[E]]], propersubclass[H[RUSSELL], RUSSELL]] // assert
```

```
True
```

```
Map[assert[implies[not[empty[fix[E]]], #]] &,
  SubstTest[propersubclass, P[x], P[y], {x → H[RUSSELL], y → RUSSELL}]] // Reverse
```

```
or[equal[0, fix[E]], not[subclass[P[RUSSELL], H[RUSSELL]]]] == True
```

Because the statement that **P[x]** is a proper subclass of **P[y]** is simplified by rewrite rules, it is evident that this continues infinitely.

```
propersubclass[P[x], P[y]]
```

```
and[not[subclass[y, x]], subclass[x, y]]
```

```
Map[assert[implies[not[empty[fix[E]]], #]] &,
  SubstTest[propersubclass, P[P[x]], P[P[y]], {x → H[RUSSELL], y → RUSSELL}]] // Reverse
```

```
or[equal[0, fix[E]], not[subclass[P[P[RUSSELL]], H[RUSSELL]]]] == True
```

In short, the Russell class and its string of iterated power classes are only defined by which sets are self-containing. Because these classes only exclude sets with self-membership in some form, they clearly contain all sets that have no affiliation with self-membership. So, with respect to regularity, any regular set is also in this hierarchy.

REGULAR ⊂ H[RUSSELL] ⊂ ... ⊂ P[P[P[RUSSELL]]] ⊂ P[P[RUSSELL]] ⊂ P[RUSSELL] ⊂ RUSSELL

```
subclass[REGULAR, H[RUSSELL]]
```

```
True
```

```
subclass[REGULAR, P[P[RUSSELL]]]
```

```
True
```

```
subclass[REGULAR, P[RUSSELL]]
```

```
True
```

```
subclass[REGULAR, RUSSELL]
```

```
True
```

Cyclic Membership

There are other ways a set can refute the Axiom of Regularity without belonging to itself. It could be that a set does not belong to itself, but belongs to one of its members, or to one of its member's members, etc. If this happens, then the set would belong to **fix[composite[E,E]]** or **fix[composite[E,E,E]]**. The union of a relation and its iterated composites **composite[x, x]**, **composite[x, x, x]**, etc. is the transitive closure of the relation, denoted **trv[x]**. Because the fixed point class preserves unions, the class of all **x** with any loop is the fixed point class of **trv[E]**.

```
trv[E]
```

```
composite[inverse[TC], E]
```

Instead of a set directly belonging to itself or a member of its transitive closure, one of the members of a set's transitive closure could belong to itself. The negation of this possibility is a necessary condition for membership in **H[RUSSELL]**.

```
exists[y, and[member[y, tc[x]], member[y, y]]] // assert
```

```
not[subclass[tc[x], RUSSELL]]
```

```
member[x, H[RUSSELL]]
```

```
and[member[x, RUSSELL], subclass[tc[x], RUSSELL]]
```

Another possibility for irregularity is a member of a set's transitive closure belongs to **fix[trv[E]]**.

```
class[x, exists[y, and[member[y, tc[x]], member[y, fix[trv[E]]]]]]
```

```
complement[P[H[complement[fix[composite[inverse[E], TC]]]]]]
```

All regular sets do not have this property.

```
subclass[REGULAR, P[H[complement[fix[composite[inverse[E], TC]]]]]]
```

```
True
```

H[RUSSELL] does not rule out the possibility of some cyclic membership; however, the **REGULAR** class has no cyclic membership.So, **REGULAR** is a proper subclass of **H[RUSSELL]** as long as there exists a set with cyclic membership (cycle > 1).

```
implies[and[member[x, tc[x]], subclass[tc[x], RUSSELL]], member[x, H[RUSSELL]]]
```

```
True
```

```
implies[and[member[x, tc[x]], subclass[tc[x], RUSSELL]], not[member[x, REGULAR]]]
```

```
True
```

If this is the case, then there exist sets that are cyclic with respect to the membership relation.These sets with cyclic membership could be in **H[RUSSELL]** and the string of iterated power classes of the Russell class, but are not members of **REGULAR**.

FUND

Another class that arouses interest when the Axiom of Regularity is denied is **FUND**, the class of all sets that are **WELL-FOUNDED** with respect to the membership relation.

```
class[x, WELLFOUNDED[restrict[E, V, x]]]
```

```
FUND
```

How is **FUND** related to other proper classes? **FUND** is a subclass of **RUSSELL** and **P[RUSSELL]**, but **FUND** neither contains nor is contained in the rest of the **RUSSELL** power string or **H[RUSSELL**].

```
subclass[FUND, RUSSELL]
```

```
True
```

```
subclass[FUND, P[RUSSELL]]
```

```
True
```

An interesting fact about **FUND** is that **REGULAR** contains the full members of **FUND**, but is a subclass of **FUND** itself.

```
subclass[REGULAR, FUND]
```

```
True
```

```
subclass[intersection[FUND, FULL], REGULAR]
```

```
True
```

An open conjecture about **FUND** is whether the equality of **H[FUND]** and **REGULAR** implies the Axiom of Regularity. Currently, it is only known that **REGULAR** is a subclass of **H[FUND]**.

```
equal[H[FUND], REGULAR] // assert
```

```
subclass[H[FUND], REGULAR]
```

For further reading, consult Peter Aczel's book Non-Well-Founded Sets. Aczel used graph theory to explain the behavior of irregular sets. To avoid dealing with issues involving the existence of certain irregular sets, Aczel introduces the Axiom of Anti-Foundation, which states that all non-well-founded sets exist. His work was enlightening for the comprehension of such sets, but because he used decorated graphs, a concept not yet developed very far in the **GOEDEL** program, it is difficult to formulate some of his ideas.

Minimal Elements

Just as a regular set is one that does not have infinite regress with respect to the membership relation, a well-founded relation is one that has no infinite regress. So, many of the properties of regular sets can be generalized to the theory of well-founded relations. To define what a well-founded relation is more rigorously, we must introduce the concept of minimal elements. Minimal elements of a relation **R** are defined as sets **y** such that for all **x** not equal to **y**, the **pair[x, y]** is not in **R**.

```
class[y, not[exists[x, and[not[equal[x, y]], member[pair[x, y], R]]]]
```

```
complement[fix[composite[R, Di]]]
```

```
minimal[R, V]
```

```
complement[fix[composite[R, Di]]]
```

The class of minimal elements of a given relation have various properties. The class of all minimal elements of a relation contains the complement of its range, but is a subclass of the union of the complement of its range and the fix point set of the relation. In other words, all sets not in a relation's range are minimal; however, members of a function's fixed point set may or may not be minimal.

```
subclass[complement[range[R]], minimal[R, V]]
```

```
True
```

```
subclass[minimal[R, V], union[complement[range[R]], fix[R]]]
```

```
True
```

The reason that a member **y** of the fix point class of a relation **R** may or may not be a minimal element of **R** is because there may be an **x** such that **pair[x,y]** is in **R**, causing **y** not to be a minimal element. If the fixed point class of a relation is empty or if **R** is the inverse of a function, then an equality involving the minimal elements of **R** exists.

```
implies[empty[fix[R]], equal[minimal[R, V], complement[range[R]]]]
```

```
True
```

```
implies[FUNCTION[inverse[R]], equal[minimal[R, V], union[complement[range[R]], fix[R]]]]
```

```
True
```

A related concept to minimal is maximal, defined as the class of all sets **x** such that no other set **y** exists where **pair[x,y]** belongs to **R**. Because of their similar definitions, any theorem regarding minimal has an analogous theorem involving maximal.

```
equal[class[x, not[exists[y, and[not[equal[x, y]], member[pair[x, y], R]]]]], maximal[R, V]]
```

```
True
```

Well-founded Relations

Johnstone defines a well-founded relation **R** as one such that all non-empty sets have a minimal element with respect to **R**.

```
forall[x, implies[not[empty[x]], exists[y, and[member[y, x],
     forall[z, implies[member[z, x], not[member[pair[z, y], R]]]]]]]] // assert
```

```
WELLFOUNDED[composite[Id, R]]
```

In a similar way to how the class of all sets with infinite regress with respect to the membership relation is called **DESCENDING**, the class of all sets with infinite regress with respect to relation **R** is called **subvar[R]**.

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

```
subvar[R]
```

```
subvar[E]
```

```
DESCENDING
```

Analogously to how how the statement that **U[DESCENDING]]** is empty is equivanent to the Axiom of Regularity, the statement that **U[subvar[R]**] is empty is equivalent to the relational part of **R** being well-founded.

```
empty[U[subvar[E]]]
```

```
AxReg
```

```
empty[U[subvar[R]]]
```

```
WELLFOUNDED[composite[Id, R]]
```

So, clearly **R** has no self-related members if **R** is well-founded. One result of this is that the class of all minimal elements of a well-founded relation is the complement of its range.

```
implies[WELLFOUNDED[R], not[exists[x, member[pair[x, x], R]]] // assert // not // not
```

```
True
```

```
implies[WELLFOUNDED[R], equal[minimal[R, V], complement[range[R]]]]
```

```
or[equal[fix[composite[R, Di]], range[R]], not[WELLFOUNDED[R]]]
```

Thus, combined with the result above, all non-empty well-founded relations whose inverse is thin (meaning that for all **y**, the class of all **x** such that the **pair[x,y]** is in **R** is a set) must not have its domain contained in its range.

```
implies[and[not[empty[R]], WELLFOUNDED[R], thin[inverse[R]]],
 not[subclass[domain[R], range[R]]]]
```

```
True
```

Recursion and Induction

When one has a well-founded relation, it allows recursion and induction to be performed. Well-founded recursion on relation **R** builds a function of one variable **F** from a function of two variables **G** such that **APPLY[F, x]** is equal to **APPLY[G, PAIR[x, composite[F, id[image[R, set[x]]]]]]**. The following statement says this more precisely, where **F** = **rec[G,R]**.

```
implies[and[FUNCTION[G], equal[domain[G], cart[V, V]],
  thin[R], WELLFOUNDED[inverse[R]], member[x, V]],
 equal[APPLY[rec[G, R], x], APPLY[G, PAIR[x, composite[rec[G, R], id[image[R, set[x]]]]]]]]
```

```
True
```

The restriction **composite[F, id[image[R, set[x]]]]** is that part of **F** that has been built up "before" **x** with respect to relation **R** starting from minimal elements of **inverse[R]**. Well-founded induction is a similar process, but instead of evaluating a function at each set, a statement of truth is proven, and the only thing needed for proving the truth of x is the truth of its **R**-predecessors, not the truth of its entire history. Well-founded induction has yet to be proven in the **GOEDEL** program; however, some special cases have been such as epsilon induction.

```
implies[and[AxReg, assert[forall[x,
    implies[forall[y, implies[member[y, x], member[y, p]]], member[x, p]]]]], equal[p, V]]
```

```
True
```

The common mathematical induction comes from this well-founded induction of the successor function on the class of natural numbers, as one starts with the base case (minimal elements), and goes on to prove the statement for the entire class.

```
implies[and[member[0, p], forall[x, implies[member[x, p], member[succ[x], p]]]],
  subclass[omega, p]] // assert
```

```
True
```

Zorn's Lemma

A partial ordering **R** is a transitive, reflexive, anti-symmetric relation. One says that the fixed point class **fix[R]** is partially ordered by **R**. Because of these properties, stating that the **pair[x,y]** is a member of a partial ordering is often shortened to **x≤y**. A member **x** of a subclass of **fix[R]** is said to be maximal if there are no members of that subclass greater than **x**. A **chain** is a totally ordered class for a partial ordering, meaning for all pairs of distinct members of a chain, one is "greater" than the other with respect to the partial ordering.

```
class[t, forall[x, y,
  implies[and[member[x, t], member[y, t]], or[member[pair[x, y], R], member[pair[y, x], R]]]]]
```

```
chains[R]
```

A chain has an **upper bound** if there exists an element of the poset that is greater than or equal to each element of the chain.

```
class[z, exists[m, forall[t, implies[member[t, z], member[pair[t, m], R]]]]]
```

```
domain[UB[R]]
```

Another axiom of set theory with a plethora of consequences is the Axiom of Choice, which states that for any collection of classes, a single element can be selected from each set, forming a choice function. A well known result in set theory is that the

axiom of choice is equivalent to Zorn's lemma, a general version of which was proven this summer. Zorn's lemma states that, any non empty poset whose chains have upper bounds has a maximal element.

```
implies[and[axch, not[equal[x, 0]], member[x, PO], subclass[chains[x], domain[UB[x]]]],
 not[empty[funpart[x]]]]
```

```
True
```

This general form of Zorn's Lemma was proven using the existing special case of restrictions of the subset relation **S**, and an isomorphism of any partial ordering with such a restriction.

```
implies[and[axch, member[x, V], subclass[Uchains[x], image[inverse[S], x]]],
 not[subclass[x, image[inverse[PS], x]]]]
```

```
True
```

Classical results in set theory prove the well ordering theorem from Zorn's lemma; however, this has yet to be proven in the **GOEDEL** program. The obstacle here is that Zorn's lemma only applies to sets and not proper classes, and for the well-ordering theorem, the hard part is to show that the class of ordinals that are equipollent to a set is again a set.

Notebooks

Here is a list of the notebooks Professor Belinfante and I worked on over the summer, along with a brief description of each. All of these notebooks can be found in the co-work section of http://www.math.gatech.edu/~belinfan/research/autoreas/goedel/nb/.

chn-cond.nb: This notebook proves a key step in the proof of Zorn's lemma. It shows how the Uchains condition of the already known theorem can be changed to better resemble the chains condition in the general form.

cond-fp.nb: Several rewrite rules involving fixed point classes. A conditional rewrite rule is proven for **fix[composite[x,y]]** when **x** is a subclass of the inverse of **y**. This notebook also includes theorems about minimal and maximal elements of well-founded relations.

max-funp.nb: This notebook outlines the final steps of the proof of Zorn's lemma. First, the notebook shows the equivalence between the statement that a partial ordering has a maximal element to a statement about the functional part of a composite relation that includes **funpart**. Then, a proof is given showing how the functional part of the composite statement translates to the functional part of the partial ordering being non-empty, finishing the proof of Zorn's lemma.

min-memb.nb: A few rewrite rules relating the concepts of minimal and maximal. In short, a **pair[x,y]** is in **MINIMAL[z]** iff **y** is a minimal element of the restriction of **z** to set **x**. An analogous rule applies to **MAXIMAL[z]**.

pc-rus.nb: This is a proof showing how, in the absence of the Axiom of Regularity, **P[RUSSELL]** can be a proper subclass of the **RUSSELL** class by providing an appropriate counter-example to the equality between the two classes.

regress.nb: A well-formulated proof of how the Axiom of Regularity denies the possibility of infinite regress by representing a set with infinite regress as the range of a function that maps natural numbers to sets.

u-fp-di.nb: Two concise theorems involving minimal and maximal elements of a relation. These prove the union of the fixed point set of a relation with the complement of its minimal members is its domain. An analogous theorem is proven involving range and maximal.

Works Cited

Aczel, Peter. *Non-Well-Founded Sets*. Center for the Study of Language and Information, Stanford, CA, 1988.

Gödel, Kurt. *The Consistency of the Axiom of Choice and of the Generalized Continuum Hypothesis*, Princeton University Press, Princeton, NJ, 1940.

Johnstone, P.T. *Notes on Logic and Set Theory*. Cambridge University Press, Cambridge, 1987.

Quaife, Art. *Automated Development of Fundamental Mathematical Theories*. Dordrecht ; Kluwer Academic, 1992.