Automated Discovery of Inductive Theorems

Keywords: theorem proving and **knowledge acquisition**

Abstract

Inductive mathematical theorems have, as a rule, historically been quite difficult to prove – both for mathematics students and for automated theorem provers. That said, there has been considerable progress over the past several years, within the automated reasoning community, towards proving some of these theorems. However, little work has been done thus far towards automatically discovering them. In this paper we present our methods of discovering (as well as proving) inductive theorems, within an automated system. These methods have been tested over the natural numbers, with regards to addition and multiplication, as well as to exponents of group elements.

1 Introduction

There have been considerable advances made over the past fifty years in automated theorem proving, including the proving of inductive theorems. However, regarding automated theorem *discovery* (theorems of any kind), relatively little has been published, other than works such as [Lenat, 1982; Colton, 2002; Gao, 2004; McCasland *et al.*, 2006]. Moreover, to our knowledge, there is as yet nothing in the literature about automated discovery of inductive theorems.

In this paper we briefly describe some of our methods for discovering (as well as proving) inductive theorems, within an automated system. We have tested these methods over two different representations of the natural numbers, each with respect to the operations of addition and multiplication. In addition, we have applied these methods to basic group theory, regarding (natural number) exponents of group elements. Admittedly, our case studies (see Section 5 and Appendix A) have thus far been rather limited; in particular, they have only concerned theorems proven from (what many mathematicians refer to as) the First Principle of Mathematical Induction. Nevertheless, the results are quite promising. In particular, we have obtained the usual associativity, commutativity and distributivity theorems for both of the aforementioned representations of natural numbers, and the usual theorems regarding exponents (in group theory).

2 The Main Idea

We should perhaps, at this point, state that we do not claim to have found, nor indeed, do we ever expect to find, a single approach for automatically discovering every inductive theorem that one could want. Instead, we offer an approach for discovering a significant number of the more "routine" inductive theorems – particularly theorems involving equality. For the moment, we only give a rough sketch of our main idea, which actually is really quite simple. The implementation of our ideas, however, is arguably not so simple. Details will follow.

The key to our approach is to first find an "interesting" proposition $P(n)$ that holds for the "2-case" – bearing in mind, of course, that "2" might well have any one of several representations, depending, in part, on the recursive datastructure. (We will henceforth use $\textsc{two}$ to represent the generic "2"). Once $P(\textsc{two})$ is established, the remaining steps obviously are guided by the appropriate induction axiom. That is, determine whether $P(base)$ is also true (where $base$ runs through the list of base elements), and if so, then apply the step case(s) of the induction axiom – e.g., try to prove that $P(k) \implies P(k+1)$. If all this succeeds, and importantly, if $P(n)$ is determined to be non-trivial, then $P(n)$ (for arbitrary n) is added to the database as a Theorem (by which we mean the sort of result that mathematicians would call a theorem, lemma, corollary, etc.). If at any point this process fails, then backtrack, and try again.

We choose to focus initially on the $\textsc{two}$-case, because it seems, on the whole, to be optimal, in terms of predicting non-trivial results, whilst keeping the search space relatively small. As for the 1-case, there are too many statements $P(1)$ that are true, for which the corresponding $P(n)$ is not true in general. One could instead try the 3-case (or any $k > 2$), but this usually results in a much larger search space, whereas any additional rewards, for having gone beyond $\textsc{two}$, are typically minimal.

There are, admittedly, some considerable limitations to this method; in particular, it will almost certainly not find the well-known formula of Gauss, that $1+\cdots+n = n(n+1)/2$. (For this, and similar theorems to be discovered, one might hope that an examples-based approach, perhaps akin to that used by Colton [2002], would work). That said, this method has, in our case studies to date, succeeded in discovering (and proving) all the inductive Theorems that we were expecting/wanting it to find.

3 Finding a TWO-Case

Before we proceed further, we remark that, while our work thus far only involves inducting over the natural numbers, we shall endeavour to present our methods in a rather more general context. Thus, one might think of the TWO-case as a proposition involving, for example, a list of length two. One ought, however, to make allowances for the likelihood that some of our ideas will not (easily) be adaptable to certain settings. That said, we hope that these ideas might provide sufficient stimulation for others to likewise pursue the automatic discovery of inductive theorems.

3.1 Finding an Appropriate Induction Axiom

In our system[1], we build theories in layers, in essentially the same manner as is done by mathematicians, and more or less in keeping with the Little Theories approach (see [Farmer *et al.*, 1992]). This of course implies that, whilst investigating one theory, the appropriate induction axiom might reside in a separate theory. Indeed, this is the case in our group theory example.

Nevertheless, for our system, this does not present any real difficulty, since the appropriate induction axiom is rather easily identifiable from the functions/constants in the theory being investigated. This is so, in part because we require unique names for all axioms, functions and constants.

We should point out that while the user-provided axiom is required by the system to determine the nature of the base and step cases, in addition to the value of TWO, we actually employ what is usually called "Structural induction", in order to produce a (proven) Theorem.

3.2 Finding TWO

In order to find an interesting TWO-case, one ought to first find a useful representation for TWO. Even though this, along with most everything else, is to be done as automatically as possible, the induction axiom provides most of the necessary information. Clearly, TWO is either one or two steps up from the (last) base element (depending on whether "1" or "0" is that element), where "step" is of course determined by the step case(s). For example, in one of our case studies, where the representation of $\mathbb{N}$ (the natural numbers) is given by the Peano Postulates, we have 0 as the base element. In our other chosen representation (as found, for example, in [Dodge, 1975]) of the naturals, which we denote by $\mathbb{N}^*$, the base is 1. It must be said that it is not all that crucial to get TWO right – "3" will do, but at a cost. (For this reason, we leave it to the reader to determine how best to automatically detect whether "0" or "1" is the base element – but obviously, the name given to this element is quite irrelevant. What *is* relevant, is the set of properties it possesses. Suffice it to say that our means of determining this is tantamount to determining what "addition" is, and whether the base element is an identity for this "addition".) That said, for our purposes, it is important to use the step representation of TWO. That is, in $\mathbb{N}$, TWO takes the form $s(s(0))$, whereas in $\mathbb{N}^*$, it looks like $1+1$.

[1]For reasons of anonymity, we do not, for the time being, provide the name of our system.

3.3 Generating Terms

As mentioned previously, we are at present primarily interested in Theorems about equalities. Thus, for the remainder of this section, we shall restrict our focus to equational propositions. This is, admittedly, a considerable restriction; nevertheless, it still allows the possibility of significant achievements. Equations do, after all, represent a sizable and important portion of mathematical theorems. Moreover, given that our methods only assume the reflexive, symmetric and transitive properties of equality, one could quite reasonably expect these methods to apply as well to any other equivalence relation, though we have not yet tested them on such.

Moving on, once our system has completed the above tasks, it begins generating a sequence of terms (together with appropriate hypotheses, such as $a, b \in \mathbb{N}$, etc.), each of which potentially represents the left-hand side of an equation (Theorem). This sequence generation is not random – it is based on an analysis of the given axioms, and an assumption that for certain situations, one looks for certain properties. For example, given a binary operation, one is likely to be interested in the associativity and commutativity properties, and given a pair of these operations, the distributive property – provided that distributivity makes sense.

We remark that in a future version of our system, there should be much greater flexibility here, since these "properties of interest" will be dictated more by the axioms provided by the user, rather than by a set programme. Since the term-generator is due to change, we are a bit reluctant to go into great detail about precisely how these terms are at present generated. We will, however, give a rough idea of our plans for the future term-generator, and the reader should understand that the present term-generator is based on a rather restricted version of these plans.

The idea is to collect, for each operation/function to be investigated, the equational axioms/definitions for all of the properties that are relevant/applicable, either to one of these operations/functions, or to a combination (preferably having no more than two distinct ones) of them – at least, the combinations that make sense. The term-generator should then produce the left (or perhaps the right) hand side of the relevant equation in each of the collected axioms/definitions, as adapted to the appropriate setting. For example, suppose that the operations $+$ and $\cdot$, and the properties of associativity, commutativity, and distributivity (both left-hand and right-hand), have all previously been defined. In this case, the term-generator should produce the terms: $a+(b+c)$, $a\cdot(b\cdot c)$, $a+b$, $a\cdot b$, $a\cdot(b+c)$, $(b+c)\cdot a$, $a+(b\cdot c)$ and $(b\cdot c)+a$, where a, b and c belong to the appropriate set. Moreover, if a constant C has been declared, then the term-generator should also produce terms in which each relevant operation/function has been applied to C (along with however many variables are needed). For example, given the constant 0, then the term-generator should also produce the terms: $a+0$, $0+a$, $a\cdot 0$ and $0\cdot a$.

Note that, in some cases, not all of the terms mentioned above need be investigated. For instance, included in our axioms for $\mathbb{N}$ (see Appendix A.1), is the statement that $a+0=a$

(for any $a \in \mathbb{N}$). One would not, therefore, expect to find any Theorem specifically about $a + 0$, and thus, this term can be discarded. On the other hand, axioms which involve a combination of a (binary) operation with a (unary) function, offer an opportunity for further exploration. Note that axiom 8 (respectively, 10) in Appendix A.1, combines $+$ (respectively, $\cdot$) with the successor function. Comparing 8 with axiom 7 (respectively, 9), the successor function effectively replaces 0. This begs the question, what if the variable inside the successor function were replaced with 0? (Note that axiom 13 in Appendix A.3 has a similar combination of the exponent operation and the successor function). Hence, our term-generator of the future will (as the present version already does, to some extent) take into account all the relevant axioms and Theorems in the database, in determining which terms to investigate.

The reader can see most of the terms generated in our studies, as the successful ones (in terms of producing a Theorem, that is) appear as the left-hand side in the inductive Theorems found in Section 5. Note that each term involves one or more operations and/or other functions described in the axioms, along with an appropriate number (at least one) of "fixed, but arbitrary" variables (to borrow a mathematicians' expression), and occasionally includes one or more constants. One such term that does not appear below, although it was generated by our system, is the term $(a * b)^n$, where $a, b \in G$ and $n \in \mathbb{N}$ (see Section 5.3). Since the group in question is not necessarily abelian, then there is essentially nothing that can be proven about this term. Indeed, our system tries to find something interesting to prove in this case, but gives up (as it should) after a few seconds.

This last example helps point out that the ordering of this sequence of terms is not, contrary to what one might think, all that critical. We agree that, in order to prove certain inductive theorems, it is important (if not essential) to already have certain lemmas at hand. And indeed, our system would not be able to prove, for example, commutativity of multiplication, without having discovered most of the previously found Theorems. Nevertheless, if the given ordering of the sequence of terms does not produce, for example, this commutativity result, then one may simply run the system again (and perhaps repeatedly), until all the necessary lemmas have been found.

We should point out that not all of the Theorems found by our system are inductive – one ought not expect them to be. The reader should understand that the process for generating these non-inductive Theorems is quite different from what we have been describing here; enough so that another paper[2] is required to fully explain it. Suffice it to say that the non-inductive-type process, like the inductive one, is automatic.

3.4 Finding an Interesting Case

For each generated term t, one of the variables (call it v) in t is chosen as the induction variable, and is replaced throughout the term (allowing for more than one occurrence of this variable in t) by TWO. The system then uses forward chaining, applying whatever axioms and Theorems are at hand, to find another term t', different from t, but such that t' contains TWO and $t(\text{TWO}) = t'(\text{TWO})$. As stated earlier, once such a term t' is found, then the system attempts to prove that $t(base) = t'(base)$, and if this succeeds, then it tries to prove the appropriate step case(s), as determined by the relevant induction axiom. (We shall have more to say about the step case, and its proof, in the next section). Even should all this succeed, there is still one more hurdle to clear, before this result is declared to be a Theorem; namely, that $t(v) = t'(v)$ should be a non-trivial result. As indicated earlier, if any stage should fail, then the system backtracks, including to the point where v was chosen.

As for precisely what is meant by "non-trivial" (and for that matter, "trivial"), this is an interesting study in itself, and is well beyond the scope of this paper. We tend to equate "trivial" with what [McCasland *et al.*, 2006] refer to as "already-known". In more practical terms, for our purposes, "trivial" effectively means, that which the system can prove, by using only a specific, limited subset of prescribed procedures and Theorems, along with all the given axioms. The exact makeup of this specific subset can, and perhaps should, vary, depending upon one's objectives. For instance, if the system's main goal is to emulate, as much as possible, the human mathematical process, then "trivial" should mean what mathematicians think it means. Alternatively, if the system is trying to automatically discover lemmas that might prove useful for an automated theorem prover (ATP), then "trivial" might well mean something quite different. (In future work, we hope to combine our system with various ATP's, for this very purpose. Hence, we intend to parameterize our specification of "trivial", in order to make it more adaptable to this, and other situations).

Leaving further discussion of trivialities aside, we return our attention to finding a suitable term t'. As we suggested earlier, it is important that we use the step representation of TWO. One reason for this is that otherwise, the system might not be able to make best use of the necessary axioms, required to find t'. Case in point, consider the associativity of addition in $\mathbb{N}^*$. In this situation, $\text{TWO} = 1 + 1$, and $t(\text{TWO}) = (a + b) + (1 + 1)$. By repeated application of Axiom 5 (see Appendix A.2), one can obtain (by hand) the following string of equalities:

$$\begin{aligned}(a+b)+(1+1) &= ((a+b)+1)+1 \\ &= (a+(b+1))+1 \\ &= a+((b+1)+1) \\ &= a+(b+(1+1)).\end{aligned}$$

Observe that the last term is the only other term in the sequence (besides t) that contains TWO, and thus is the term we seek. Note that, had we replaced TWO with 2, say, then we would have been stuck. Note also, that the above string of equalities effectively shows the way for proving the step case – another (potential) reason for using the step representation of TWO. This last observation, however, will not necessarily hold in all situations. Hence, we have implemented other means of proving the step case, which we will describe in some detail later.

Not surprisingly, our system does not typically arrive at the term t' nearly so easily as might be suggested by the above

[2]We indeed have written such a paper, but again, for reasons of anonymity, we are unable to provide a reference here.

equations. Indeed, particularly whenever there is an identity involved, the search space can be literally overwhelming. One method we use to limit this search space, is to put a cap C on the size of allowable terms, as measured by the following variant (denoted $m(t)$) on the standard size measurement of terms:

$$m(v) ::= 0; \text{ where } v \text{ is a variable or constant}$$

$$m(f(s_1, \ldots, s_n)) ::= 1 + \sum_{i=1}^{n} m(s_i); \text{ where } f \text{ is a function .}$$

(Note that each term t is quantifier-free, because we rely on fixed, but arbitrary variables). This cap is set at

$$C ::= M + m(\text{TWO}) + 2,$$

where M is initially set at

$$M ::= m(t).$$

The extra cushion of $m(\text{TWO}) + 2$ is allowed, in order to accommodate such results as distributivity, wherein the sought-after right-hand side might be larger than the given left-hand side, and moreover, the induction variable (and hence, TWO) might appear twice.

As for the search itself, we have found that a two-stage approach works quite well. In the first stage, we limit the reach of the forward chaining process, in much the same way as discussed previously, regarding "trivialities". In particular, we collect all "promising" terms s, reachable (subject to our imposed limitations) from t, such that $t(\text{TWO}) = s$ and $m(s) \leq C$. If this does not produce the desired term t', then we add to our collection whatever "promising" terms can be reached from each of the terms s already in our collection, and so on. Along the way, each of these "promising" terms is sent to the second stage, which uses a directed (but still limited) search to see if it is "close to" a term s' that contains TWO. That is, can a term s' be (quickly) found that contains TWO and such that $s'(\text{TWO}) = s$. Moreover, for any promising term s, if $m(s) < M$, then M is reset to $M ::= m(s)$, further restricting the search space.

In the above associativity example, this second stage finds that the term $(a + (b+1)) + 1$ is indeed "close to" the desired term $a + (b + (1+1))$, and in effect bypasses the last but one term.

The process continues in this vein, until either a suitable term t' is found, or no more promising terms s can be found. Note that the transitivity of equals insures the soundness of this approach.

4 Proving the Step Case

Once we reach the stage of trying to prove the step case, we could hand the proof off to one of the ATP's that are designed to handle induction proofs. However, our design philosophy, together with a strong desire to keep everything in-house, required that we built our own equation-prover. Besides, there is still the question of whether the resulting general equation is non-trivial, and ATP's, quite understandably, were simply not designed to answer this question. We admit that our own equation-prover could stand some improvement; nevertheless, it has, thus far, succeeded in proving everything we wanted it to.

We built the induction-proof portion of our equation-prover on the (fairly obvious) assumption that, in order to prove the "$k+1$-case", one is almost certainly going to make use of the knowledge provided in the "k-case". Note that, in our situation, both cases are, in fact, equations. Hence, both the "k-equation" and the "$k+1$-equation" are passed to the equation-prover. Taking the left-hand side of the "$k+1$-equation", which we will denote by $lhs(k+1)$, the prover first uses a process much like the second stage process described above, in order to see if this term is "close to" a term t that contains $lhs(k)$. If so, then $rhs(k)$ (i.e., the right-hand side of the "k-equation") is substituted in the appropriate place in t (provided that substitution is allowable), and an attempt is made to prove that the resulting term equals $rhs(k+1)$. Should all this succeed, then, of course, the proof is complete.

If each step in this process succeeds quickly, then well and good. If not, then we use a piece-wise search, which is certainly reminiscent of, but somewhat different from, rippling (see, for example, [Bundy *et al.*, 2005]). Here, the target term (e.g., $lhs(k)$) is broken down into subterms (to begin with, just one level down, in terms of the tree structure). A search is then made for a term that is equal to the given term (e.g., $lhs(k+1)$), but that contains the first subterm of the target term. If this succeeds (including, of course, the case that the given term already contains this subterm), then a subsequent search is made, regarding the next subterm, with the proviso that the previous subterm not be lost. If any search fails, then the relevant subterm is broken down into its subterms, and we proceed as before. These searches continue, until the desired target term is found, or until all searches fail.

Should this still fail, then the prover begins again, but starting from the respective right-hand sides, moving to the left. Should this last attempt fail, then the proof fails.

As an example of the piece-wise search, consider the (left-hand) distributivity Theorem, again in $\mathbb{N}^*$. The equation-prover receives the (assumed) "k-equation"

$$a \cdot (b + k) = (a \cdot b) + (a \cdot k),$$

along with the (to-be-proved) "$k+1$-equation"

$$a \cdot (b + (k+1)) = (a \cdot b) + (a \cdot (k+1)).$$

In order to make use of the given equation, the prover needs to find a term t such that $t = a \cdot (b + (k+1))$ and t contains $a \cdot (b+k)$. It could, of course, stumble around, until it (hopefully) eventually succeeded. However, using the piece-wise search, it successively searches for terms t_1 and t_2 such that:

$$t = t_1 \text{ and } t_1 \text{ contains } a$$
$$t_1 = t_2 \text{ and } t_2 \text{ contains } a \text{ and } b + k.$$

Clearly t already contains a, but does not contain $b+k$. Thus, a search is made for t_2, and in fairly short order, it finds that $t_2 = (a \cdot (b+k)) + a$ satisfies the requirements. Now $rhs(k)$ can be applied, which gives

$$(a \cdot (b+k)) + a = ((a \cdot b) + (a \cdot k)) + a.$$

At this point, the piece-wise search can again be used, with $rhs(k+1)$ as the target term. The sought-after subterms are $a \cdot b$, which $((a \cdot b) + (a \cdot k)) + a$ has, and $a \cdot (k+1)$, which it does not have. Once again, the search rather quickly finds the following sequence of equations:

$$\begin{aligned}((a \cdot b) + (a \cdot k)) + a &= (a \cdot b) + ((a \cdot k) + a) \\ &= (a \cdot b) + (a \cdot (k + 1)),\end{aligned}$$

and the proof is complete.

5 Theorems

We include the Theorems found by our system, for both of the aforementioned representations of the natural numbers, and for group theory. The inductive Theorems are designated by *. We remark that our system is programmed to determine, for each binary operation, whether the operation is closed. (Of course, if there is an axiom that provides this information, then nothing else need be done.) Hence, in addition to the following results, the system also discovered that, for $a, b \in N$ and for $g \in G$, then $a + b, a \cdot b \in \mathbb{N}$ and $g^n \in G$. These results were indeed added to the database, but were not recorded as "Theorems".

The code for our system is written in two languages. All of what one might consider the "mathematics", is done in Prolog; everything else is handled in Java. For these experiments, we ran our system on a Pentium 4 CPU, 2.40GHz machine, with 512MB RAM. The time taken, rounded to the nearest second, to generate each list of Theorems (including the non-inductive, as well as the inductive results) is provided at the end of each list.

For the convenience of the reader, the data here have been rewritten in standard mathematical notation.

5.1 Theorems in the Natural Numbers

The axioms, from which these Theorems (and the Theorems in the remainder of this section) are derived, can be found in Appendix A.

Assume throughout that $a, b, c \in \mathbb{N}$.

Theorems:	
1.	$a + s(0) = s(a)$
2.*	$(a + b) + c = a + (b + c)$
3.*	$0 + a = a$
4.*	$s(b) + a = s(b + a)$
5.*	$a + b = b + a$
6.	$a \cdot s(0) = a$
7.*	$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
8.*	$(b + c) \cdot a = (b \cdot a) + (c \cdot a)$
9.*	$(a \cdot b) \cdot c = a \cdot (b \cdot c)$
10.*	$0 \cdot a = 0$
11.*	$s(0) \cdot a = a$
12.*	$a \cdot b = b \cdot a$

The above list of Theorems was generated in 84 seconds.

5.2 Theorems in the Positive Naturals

Assume throughout that $a, b, c \in \mathbb{N}^*$.

Theorems:	
1.*	$(a + b) + c = a + (b + c)$
2.*	$a + 1 = 1 + a$
3.*	$a + b = b + a$
4.*	$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
5.*	$(b + c) \cdot a = (b \cdot a) + (c \cdot a)$
6.*	$(a \cdot b) \cdot c = a \cdot (b \cdot c)$
7.*	$1 \cdot a = a$
8.*	$a \cdot b = b \cdot a$

The above list of Theorems was generated in 14 seconds.

5.3 Theorems in Group Theory

Assume throughout that $a, b, c \in G$ and that $m, n \in \mathbb{N}$. Note that $inv(a)$ denotes the inverse of a, and that for this study, we do not consider negative exponents.

Theorems:	
1.	$inv(inv(a)) = a$
2.	$b * a = a \implies b = e$
3.	$a * b = a \implies b = e$
4.	$b * a = e \implies inv(a) = b$
5.	$a * b = e \implies inv(a) = b$
6.	$inv(a) * inv(b) = inv(b * a)$
7.	$c * a = c * b \implies a = b$
8.	$a * c = b * c \implies a = b$
9.	$a^{s(0)} = a$
10.*	$e^n = e$
11.*	$a^m * a^n = a^{m+n}$
12.*	$(a^m)^n = a^{m \cdot n}$

The above list of Theorems was generated in 90 seconds. Of that time, only 19 seconds were spent on the "exponent" Theorems (i.e., beginning with Theorem 9, which includes all the inductive results). This includes the discovery/proof that $a^n \in G$, along with the (laudable, but futile) attempt to find something interesting to prove about $(a * b)^n$.

5.4 Significance of These Theorems

The reader might not be fully aware of the difficulties in automatically proving some of the above Theorems. Particularly, the commutativity Theorem for multiplication in $\mathbb{N}$ has been notoriously hard for conventional induction provers to prove. One reason, several intermediate lemmas are normally required to be on hand, before trying to prove the commutativity result. Even with these lemmas provided, some provers still cannot succeed in the proof, without human intervention.

These difficulties exist, even when the provers are told what to prove! The fact that our system was not told what to prove, but had to first discover each Theorem, and then prove it, – including the commutativity Theorem – seems to be a rather significant achievement.

While there has been some work by other groups, towards automated discovery of (non-inductive) Theorems, no one else, to our knowledge, has published any work on automatically discovering inductive Theorems. Thus, there is no one to whom we can effectively compare our work.

6 Conclusions and Future Work

We have described our methods, and to some extent, our implementation of these methods, for automatically discovering and proving inductive theorems. We have tested our methods, albeit in somewhat limited fashion, and have included the results of our tests. While our work is still ongoing, the results to date are quite promising.

Besides the previous references to future work, we are quite keen to try out these methods in other theories – particularly for lists, as well as over the integers (including integer exponents of group elements). We anticipate the need for at least some (perhaps only minor) adjustments to our implementation, if not to our methods.

A Axioms

We include the axioms provided to our system, for two versions of the natural numbers and for group theory. For the convenience of the reader, the data here have been rewritten in standard mathematical notation.

A.1 The Natural Numbers

The following are the axioms/definitions provided to our system, for the natural numbers (based on the Peano Postulates).

Axioms: Given that $a, b \in \mathbb{N}$;

1. $\mathbb{N}$ is a set
2. $0 \in \mathbb{N}$
3. $s(a) \in \mathbb{N}$
4. $s(a) = s(b) \iff a = b$
5. $s(a) \neq 0$
6. If $S \subseteq \mathbb{N}$ such that:
   (i) $0 \in S$; and
   (ii) $k \in S \Rightarrow s(k) \in S$;
   then $S = \mathbb{N}$
7. $a + 0 = a$
8. $a + s(b) = s(a + b)$
9. $a \cdot 0 = 0$
10. $a \cdot s(b) = (a \cdot b) + a$

A.2 The Positive Natural Numbers

The following are the axioms/definitions provided to our system, for the positive natural numbers ($\mathbb{N}^*$) (as found, for example, in [Dodge, 1975]).

Axioms: Given that $a, b \in \mathbb{N}^*$;

1. $\mathbb{N}^*$ is a set
2. $1 \in \mathbb{N}^*$
3. $a + b \in \mathbb{N}^*$
4. $a \cdot b \in \mathbb{N}^*$
5. $(a + b) + 1 = a + (b + 1)$
6. $a \cdot 1 = a$
7. $a \cdot (b + 1) = (a \cdot b) + a$
8. If $S \subseteq \mathbb{N}^*$ such that:
   (i) $1 \in S$; and
   (ii) $k \in S \Rightarrow k + 1 \in S$;
   then $S = \mathbb{N}^*$

A.3 Group Theory

The following are the axioms/definitions provided to our system, for group theory. Note that $inv(a)$ denotes the inverse of a, and that for this study, we do not consider negative exponents.

Axioms: Given that $a, b, c \in G$ and $n \in \mathbb{N}$;

1. G is a set
2. $a * b \in G$
3. $a = b \implies c * a = c * b$
4. $a = b \implies a * c = b * c$
5. $(a * b) * c = a * (b * c)$
6. $e \in G$
7. $a * e = a$
8. $e * a = a$
9. $inv(a) \in G$
10. $inv(a) * a = e$
11. $a * inv(a) = e$
12. $a^0 = e$
13. $a^{s(n)} = a^n * a$

References

[Bundy *et al.*, 2005] A. Bundy, D. Basin, D. Hutter, and A. Ireland. *Rippling: Meta-level Guidance for Mathematical Reasoning*, volume 56 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2005.

[Colton, 2002] S Colton. *Automated Theory Formation in Pure Mathematics*. Springer-Verlag, 2002.

[Dodge, 1975] C. W. Dodge. *Numbers & Mathematics*. Prindle, Weber & Schmidt, Inc. , 2nd edition, 1975.

[Farmer *et al.*, 1992] W. M. Farmer, J. D. Guttman, and F. J. Thayer. Little theories. In D. Kapur, editor, *CADE11*, pages 567–581, 1992.

[Gao, 2004] Y Gao. Automated generation of interesting theorems. Master's thesis, University of Miami, 2004.

[Lenat, 1982] D. B. Lenat. AM: An artificial intelligence approach to discovery in mathematics as heuristic search. In *Knowledge-based systems in artificial intelligence*. McGraw Hill, 1982. Also available from Stanford as TechReport AIM 286.

[McCasland *et al.*, 2006] R. L. McCasland, A. Bundy, and P. F. Smith. Ascertaining mathematical theorems. *Electronic Notes in Theoretical Computer Science*, 151(1):21–38, 2006.