Article Submitted to Journal of Symbolic Computation

Automated Conjecture Making in Number Theory using HR, Otter and Maple

Simon Colton[1]

[1]*Department of Computing*
Imperial College of Science, Technology and Medicine
180 Queens Gate, London SW7 2AZ
United Kingdom
sgc@doc.ic.ac.uk
http://www.doc.ic.ac.uk/~sgc

Abstract

One of the main applications of computational techniques to pure mathematics has been the use of computer algebra systems to perform calculations which mathematicians cannot perform by hand. Because the data is produced within the computer algebra system, this becomes an environment for the exploration of new functions and the data produced is often analysed in order to make conjectures empirically. We add some automation to this discovery process by using the HR theory formation system to make conjectures about Maple functions supplied by the user. HR forms theories by inventing concepts, making conjectures empirically which relate the concepts, and appealing to third party theorem provers and model generators to prove/disprove the conjectures. It has been used with success in number theory, graph theory and various algebraic domains such as group theory and ring theory.

Experience has shown that HR produces too many conjectures which can be easily proven from the definitions of the functions involved. Hence, we use the Otter theorem prover to discard any theorems which can be easily proven, leaving behind the more interesting ones which are empirically plausible but not easily provable. We describe the core functionality of HR which enables it to form a theory, and the additional functionality implemented in order for HR to work with Maple functions. We present two experiments where we have applied HR's theory formation in number theory. We discuss the modes of operation for the user and provide some of the results produced in this way. We hope to show that using HR, Otter and Maple in this fashion has much potential for the advancement of computer algebra systems.

1. Introduction

There is an unfortunate dichotomy between the application of computer algebra systems (CASs) and automated theorem provers (ATPs) to pure mathematics: at the moment, the concepts dealt with routinely by computer algebra techniques are of too high a complexity to prove theorems about using automated provers. There have been some attempts to bridge the gap in order to usefully apply automated theorem proving to computer algebra, including (i) the routine proving of fairly trivial theorems such as side conditions holding when calculating integrals and (ii) a semi-automated approach, where the user is actively involved in theory exploration within the CAS and the prover is called upon at specific times during the exploration, often to deal with fairly trivial theorems [1]. Ideally, automated theorem provers would be called from within a CAS whenever the user made a conjecture about the functions they were defining. However, this will take increased sophistication in automated theorem provers and is unlikely to happen in the short term.

If we change the aim of the integration of mathematical systems to be the generation of *conjectures*, rather than theorems about the functions being explored using a CAS, then it is possible to put a positive spin on the relative differences between CAS and ATP. Rather than stating that a disadvantage of ATPs is their limited abilities with concepts of a higher complexity, we note that an advantage of ATPs is that they can be used to prove theorems from first principles, i.e., directly from the axioms of a domain. Furthermore, these theorems are less likely to be of interest to the user than those which cannot be proved by an ATP system. Therefore, in a conjecture-making context, we can use ATP systems to prune conjectures which are easily provable from the definitions of the functions, thus improving the quality of the conjectures produced.

We assume a plausible 4-step model of progress in pure mathematics:

1. Some functions are defined in a particular context
2. The functions are calculated over a set of input values
3. The input/output pairs are examined in order to highlight patterns
4. Any observed patterns are stated as conjectures and proved or disproved

We note that, in general, the second step can be automated by computer algebra systems and the fourth step can be automated by theorem provers. Automating the third step — thus providing a possible bridge between CAS and ATP — is the subject of this paper. The making of conjectures necessitates a certain amount of concept formation. This is because sophisticated conjecture making involves not only finding conjectures about the given functions, but also about closely related (and not so closely related) functions. Hence, we will also be automating the first step and closing a cycle of theory formation.

The HR program [5] is a machine learning system able to perform theory formation in domains of pure mathematics. It undertakes descriptive induction tasks by forming concepts, making conjectures empirically using the examples of

the concepts, then attempting to prove and disprove the conjectures using third party software. In the experiments described here, HR interacts with the Maple computer algebra system [25] and the Otter theorem prover [20] to make, prove and prune conjectures about some Maple functions provided by the user. While Otter and similar provers have been used to prove many difficult theorems (most notably the Robbins conjecture [23]), it is highly likely that any result provable in number theory in the 5 seconds that we give Otter will be trivially true, and hence can be discarded.

To describe how these experiments were facilitated, in §2, we present the core functionality behind HR which enables it to make conjectures. In §3 we describe the additional functionality implemented for the application of HR to the generation of conjectures about Maple functions. In §4, we describe the experiments using HR to generate conjectures about some Maple functions from number theory, and in appendices A and B, we prove two theorems that HR discovered.

2. Theory Formation in HR

Machine learning programs generally fall into one of two categories. Most common are *predictive induction* systems, which, given information about a particular concept to learn, produce plausible definitions for the concept. Less common are *descriptive induction* systems. Given similar background knowledge to their predictive counterparts, descriptive systems derive categorisations and find association rules in the data. Such systems include WARMR [12] and CLAUDIEN [11]. HR is also a descriptive induction system which is designed to make conjectures about the concepts expressed in the background knowledge it is supplied with. Much of HR's functionality was employed for the application described here, and each task that it performs can be broadly placed into of the following six classes: (i) using background information from the user (ii) inventing concepts (iii) making conjectures (iv) finding counterexamples (v) proving theorems and (vi) reporting results. We describe these sets of tasks and how HR performs them in the subsections below.

2.1. Background Information

HR forms theories about a set of *objects of interest*, which are integers in number theory, graphs in graph theory, groups in group theory, etc. It is given background information which describes the objects of interest, namely some initial concepts. As discussed in §3 and §4 below, the objects of interest in the sessions described in this paper are integers, and the background information is supplied in the form of Maple functions. HR works with both the examples and definitions of a concept. For this reason, the user must present concepts with both a full set of examples – calculated over the entire set of objects of interest – and a definition for each of the languages HR will use (for instance, the language of

the Otter theorem prover). Every concept HR produces will similarly have a set of examples and a definition expressed in multiple languages. If the user also specifies which concepts can be thought of as functions rather than predicates, HR will propagate this information and use it to more intelligently form a theory. If HR is expected to prove or disprove any conjectures which arise, then the user must also specify some axioms for the domain, expressed in the language of the prover/model generator employed (with relevant definitions matching those for the background concepts).

Background information can be given to HR as a flat file with information arranged in a first order format similar to that provided to Inductive Logic Programs such as Progol [24]. Unlike Progol, however, HR does not have an underlying Prolog interpreter, and the information in the files is purely a set of data points. A drawback to this representation scheme, especially in mathematical domains, is the fact that the information is not extendible. For instance, if we gave HR information about the integers 1 to 50, it would not be able to calculate any information about numbers greater than 50, which may cause problems. We are currently enabling HR to use a Prolog interpreter to overcome this limitation. We have also enabled HR to take Java code in its background information. This code is called every time the set of examples of a concept are required for a particular object of interest. As described in §3, this code may also contain a call to either Maple or the Gap computer algebra system [15].

2.2. Inventing Concepts

From the background information, HR uses ten production rules to generate a new concept from one (or two) old concepts. The production rules are described in more detail in [5] and [6], and we concentrate here on only four:

- The *compose* production rule composes functions using conjunction
- The *disjunct* production rule joins concepts using disjunction
- The *exists* production rule introduces existential quantification
- The *split* production rule instantiates objects

As an example construction, we suppose that HR is given the background concepts of the `isprime(n)` Maple function, which checks whether `n` is prime, and the `sigma(n)` Maple function, which calculates the sum of divisors of `n`. Using the compose production rule, HR invents the concept of pairs of integers, $[a, b]$ for which $b = sigma(a)$ and $isprime(b)$. Following this, it uses the exists production rule to define the concept of integers, a, for which there exists such a b, i.e., $[a] : \exists\ b\ (sigma(a) = b\ \&\ isprime(b))$. Hence HR has invented the concept of integers for which the sum of divisors is prime, a concept which we discuss later. This construction is represented in figure 1. We say that the *complexity* of a concept is the number of concepts (including itself) in the construction path of the concept, as explained further in [6]. Hence, the complexity of the concept depicted in figure 1 is the number of boxes, i.e., four.

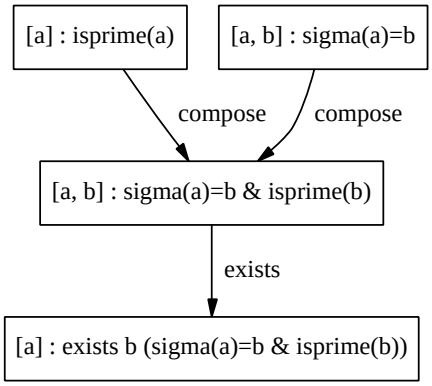

Figure 1: Construction of the concept of integers with a prime sum of divisors

2.3. Making Conjectures

HR has a number of ways to make conjectures, both by finding empirical patterns and by extracting simpler conjectures from more complex ones. Firstly, whenever HR invents a concept, it checks two things empirically:

(i) whether the concept has no examples whatsoever, in which case it makes a non-existence conjecture, i.e., that the definition of the concept is inconsistent with the axioms of the domain. For example, if HR invented the concept of square numbers which are prime, it would find no examples, and make the conjecture that none exist on the number line.

(ii) whether the concept has exactly the same examples as a previous one, in which case, it makes a conjecture that the definitions of the new and old concepts are logically equivalent. For example, if HR invented the concept of integers for which the number of divisors is 2, it would make the conjecture that the new concept is equivalent to the concept of integers which are prime.

If the concept has a non-empty set of examples which differs from all previous concepts, then the concept is new and it is added to the theory. HR also determines which concepts the new concept *empirically subsumes*, i.e., which concepts have a proper subset of the examples for the new concept. For each old concept that the new concept subsumes, HR makes the implication conjecture that the old definition implies the new definition. Similarly, HR determines which old concepts subsume the new concept, and makes the appropriate implication conjectures. From each subsumption conjecture, HR extracts implicate conjectures. For instance, if it made the implication conjecture that: $f(a)$ & $g(a) \rightarrow h(a)$ & $x(a)$, it would extract these two implicates: $f(a)$ & $g(a) \rightarrow h(a)$ and $f(a)$ & $g(a) \rightarrow x(a)$. HR extracts implicate conjectures from equivalence conjectures and non-existence conjectures in a similar fashion. For instance, if HR made the non-existence conjecture that $\nexists\ a\ (f(a)\ \&\ g(a))$,

it would extract two implicate conjectures: $f(a) \rightarrow \neg g(a)$ and $g(a) \rightarrow \neg f(a)$. We enabled HR to extract implicates, as these are often easier to comprehend than the conjectures from which they are extracted. Often, as in the case in §4, we instruct HR to discard all but the implicates. HR checks whether a new implicate has already been added to the theory, to avoid redundancy.

From implicates, HR can also extract prime implicates, which are such that no proper subset of the premises implies the goal. To do this, it tries to prove that each subset of the premises of an implicate imply the goal, starting with the singleton subsets and trying ever larger subsets. For instance, if starting with the implicate: $f(a)$ & $g(a) \rightarrow h(a)$, HR attempts to prove $f(a) \rightarrow h(a)$ and $g(a) \rightarrow h(a)$. If Otter can prove either of these conjectures, then they are added to the set of prime implicates, because clearly no proper subset of the premises imply the goal. The prime implicates represent some of the fundamental truths in a domain, so it is worthwhile extracting them.

2.4. Finding Counterexamples

The user can specify that certain objects of interest are given to HR to form a theory with, and others are held back in order to use for counterexamples. Then, whenever HR makes a conjecture, the held-back set is searched in order to find a counterexample. An advantage to this is an increase in efficiency, as often only a fraction of the objects of interest will find their way into the theory as counterexamples. Hence, whenever HR invents a concept, it will take less time to calculate the example set for the concept than if all the objects of interest were being considered. Taking this to the extreme, in experiment 1 below, we gave HR only the number 1 to start with, but we allowed it access to the numbers 2 to 30 in order to find counterexamples to false conjectures. In addition to increased efficiency, it is also instructive to look at the false conjectures HR makes for which each counterexample is introduced. In algebraic domains, HR can also use the MACE model generator [22] to find counterexamples, but discussion of this is beyond the scope of this paper.

2.5. Proving Theorems

HR has some built-in abilities to decide when a conjecture it makes is trivially true, e.g., it can tell that conjectures such as $f(a)$ & $g(a) \leftrightarrow g(a)$ & $f(a)$ are true. It also keeps a record of which concepts it generates are functions, so that it can tell that conjectures of the form $\nexists\, a\,(f(a) = k_1$ & $f(a) = k_2)$ are true if k_1 and k_2 are different ground instances. In fact, it uses its primitive theorem proving to avoid inventing concepts such as this in the first place, because it knows in advance that the concept will have no examples, leading to a dull non-existence conjecture. If HR had more sophisticated theorem proving, then we would, to a certain extent, be re-inventing the wheel, as there are many good theorem provers available for HR to use. In particular, HR invokes the Otter theorem prover to attempt to prove the conjectures it makes. HR has been interfaced to

Otter and other provers via MathWeb [14, 27], but the experiments here were undertaken using a simple file interaction. The user specifies how much time Otter is allowed for each proving attempt (usually 5 seconds).

2.6. Reporting Results

HR is able to prune the conjectures it produces and order those remaining in terms of measures of interestingness [8]. In particular, in the experiments described below, we instruct HR to keep only implicates (extracted from equivalence, non-existence and subsumption conjectures), as these are usually the easiest to understand. We also instruct HR to discard any conjectures which Otter can prove in 5 seconds, as these are likely to follow easily from the definitions of the Maple functions provided and thus be fairly uninteresting.

Of those implicates remaining, we use two measures of interestingness to order them. Firstly, each implicate comprises a concept implying a single clause, and the *applicability* of the concept gives an indication of the scope of the conjecture. The applicability of a concept is measured as the proportion of objects of interest in the theory which have non-trivial examples for the concept. The applicability of an implicate conjecture is taken as the applicability of the concept on the left hand side of the conjecture. For instance, if HR was working with the integers 1 to 30, then the concept of prime numbers would score 10/30 for applicability, because there are 10 prime numbers between 1 and 30. Hence implicate conjectures where the concept making up the premises is the concept of prime numbers will score 1/3 for applicability. Conjectures with very low applicability tend to be uninteresting, because they are usually simple re-statements of properties of individual integers (or pairs of integers). For instance, there are many conjectures about even prime numbers which can be made. However, as there is only one such even prime, these conjectures simply describe properties of the number 2. Hence, sorting the conjectures in terms of decreasing applicability can aid the user in finding the most interesting ones produced by HR.

Secondly, equivalence and subsumption conjectures relate two concepts from the theory. HR measures the *surprisingness* of these conjectures as the proportion of concepts in the construction path of either concept which are in the construction path of exactly one of the related concepts. If two concepts conjectured to be related actually share many concepts in their construction paths, their definitions are likely to be similar, and the relationship between them will probably be unsurprising, so they score poorly for surprisingness. For example, in figure 2, there are 7 concepts involved in the construction history of the conjecture relating the two concepts joined by a dotted line. Only one of these is shared by the two concepts in the conjecture, hence the conjecture scores 6/7 for surprisingness. The implicates extracted from equivalence and subsumption conjectures inherit the surprisingness value from their parent, so that these can also be measured in terms of surprisingness.

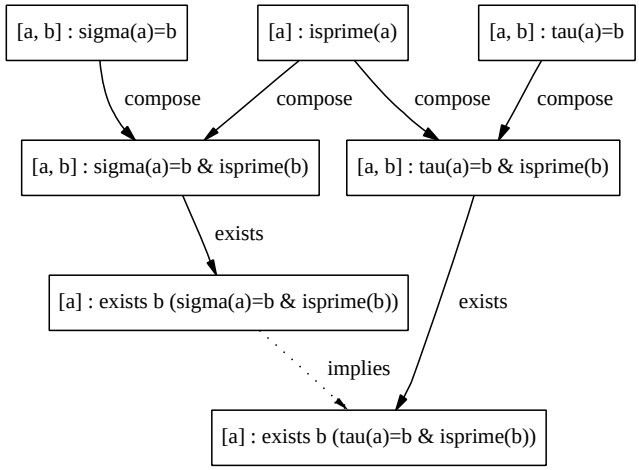

Figure 2: Construction of an implicate conjecture

3. Additional Functionality

Each novel application of HR necessitates some new functionality. In this case, we have extended HR's functionality in all the six areas discussed in §2, and we look at these improvements below. Many upgrades have been facilitated by an interpreter we have built in to HR which uses Java's reflection mechanism to interpret a subset of the full Java code specification at run-time. At present, it is able to cope with if-statements, for-loops, creation of objects and string manipulations, and we plan to enhance this. This additional functionality has not only enabled us to very much simplify HR's code in places, but has much enhanced HR's ability to search and report its findings. In addition, it has greatly improved our ability to debug HR at run time. Some other advantages gained from the interpreter are mentioned in the subsections below.

3.1. Enhancements to Taking in Background Information

As mentioned previously, HR uses an interpreter to do the calculations specified in Java by the user in the background file. It also uses this mechanism to communicate with Maple: in the Java code for the background concepts, the user can tell HR that it should invoke Maple to calculate the examples for a concept, rather than calculating them itself. HR calls Maple at the start of a session to get the initial data for the background concepts. For instance, if the user decides to start HR with the integers 1 to 10 and the Maple number theory functions of `tau(n)` and `sigma(n)`, (with `tau(n)` being the number of divisors of `n` and `sigma(n)` being the sum of divisors of `n`), then HR will use Maple to calculate `tau(1)=1`, ..., `tau(10)=4`, doing likewise for `sigma`. These functions are inbuilt, and the user only has to specify that the Maple `numtheory` package is loaded before they are called. However, the user is also able to specify a file

containing some Maple code to call a user-defined function. We take advantage of this functionality in the second experiment below. As described in [9], we have also enabled HR to extract concepts directly from Maple files, although this functionality was not used for the experiments described here.

At present, HR invokes Maple in the same way as it does with Otter, by reading a file, calling Maple in such a way that it outputs answers to another file, and then reading that file. HR, Maple and Otter are already part of the MathWeb software bus [14] and we have been successful in enabling HR to invoke Otter (and other provers) via MathWeb [27]. We can see no problem in enabling the communication between HR and Maple on a more sophisticated level via MathWeb, and we hope to do this soon.

3.2. Enhancements to Concept Formation

There have been two major enhancements to HR's concept formation. Firstly, HR calls Maple during concept formation whenever a calculation is required to fill in the examples for a new concept. For instance, if HR used its compose rule to invent the concept of `tau(sigma(n))`, then it would need to calculate `tau(sigma(10))`, which is `tau(18)=6`.

Secondly, we have improved the way in which HR writes definitions, so that the conjectures about the concepts are easier to read for the user (intended to be a mathematician). In particular, in order to make the definitions of functions which have been composed more understandable, HR collates and removes existential variables where possible. For example, when HR invents a concept with, say, the definition:

$$[a] : \exists\ b\ (f(a) = b\ \&\ \exists\ c\ (g(b) = c\ \&\ h(a) = c))$$

it first collects together the existential variables thus:

$$[a] : \exists\ b, c\ (f(a) = b\ \&\ g(b) = c\ \&\ h(a) = c),$$

then removes the existential variables b and c thus:

$$[a] : g(f(a)) = h(a)$$

It has done this by both substituting $f(a)$ for b and by removing c by equating $g(b)$ and $h(a)$. As a concrete example, HR rewrites the definition for integers with a prime sum of divisors described in §2.2 above in this way:

$$[a] : \exists\ b\ (sigma(a) = b\ \&\ isprime(b)) \quad \text{becomes} \quad [a] : isprime(sigma(a))$$

which is easier to understand. This functionality has also been useful for an application to constraint generation [10].

3.3. Enhancements to Conjecture Making

We have extended HR's functionality to enable it to make *applicability* conjectures, which state that a concept is restricted to having only a small number of examples. For instance, when HR invents the concept of integers which are equal to their number of divisors, it notices that this property is only true for integers 1 and 2. It then adds concept formation steps to the agenda which invent (a) the concept of an integer being the number 1 (b) the concept of an integer being the number 2 and (c) the concept of an integer being either 1 or 2. We call such concepts *instantiation* concepts, as they are basically the instantiation of a single object of interest (or a disjunction of similar instantiations). Having invented concept (c) using the disjunct production rule, HR then makes the conjecture that an integer is equal to its number of divisors if and only if it is equal to 1 or 2. HR is then able to identify the conjectures which involve instantiation concepts and discard them, as they are, in general, not particularly interesting.

Making applicability conjectures is part of a general tendency towards "reactive searches" during theory formation. These are heuristic searches where the user supplies certain scripts (interpreted by the Java interpreter) which specify how HR should react to certain events during theory formation. In the case of applicability conjectures, the user has informed HR that it should make such conjectures whenever a concept with low applicability is invented. We are currently experimenting by writing more complicated scripts and have found them very useful in some bioinformatics applications [4].

3.4. Enhancements to Theorem Proving

We have given HR the ability to pass to Otter the values calculated by Maple for the background functions. For example, in §4, we describe a session with HR using the Maple `tau(n)` function. During that session, HR makes instantiations, so it eventually discovers conjectures such as $\forall\ a, ((a = 1 \vee a = 2) \rightarrow tau(a) = a)$. As HR uses Maple to calculate ground instances such as `tau(1) = 1`, `tau(2) = 2,` etc., and HR gives Otter these ground instances, Otter is able to prove the above theorem and HR discards it because it is unlikely to be interesting. Note also that for concepts of arity 1, e.g., number types such as square numbers, HR also tells Otter which numbers *do not* have the property. So, for instance, HR passes `-issquare(2)` to Otter as an axiom.

Furthermore, the user is now able to act as a theorem prover and tell HR that certain conjectures are true and should be given to Otter as additional axioms for future proof attempts. For instance, in the session described in experiment 1 below, HR identifies the conjecture that $isprime(n) \rightarrow tau(n) = 2$. This follows from the definitions, and we told HR to use this as an axiom of the domain. With that information, it was able to prove many more theorems. This mode of operation makes it possible for the user to avoid specifying the axioms of the domain in advance, as HR will re-discover (some of) them. In fact, in experiment 1 below, we gave HR no axioms of number theory in advance and

relied upon it finding them for us. This mode of working is useful for generating results immediately, but can lead to much user intervention. Alternatively, as in experiment 2 below, the user can supply axioms in advance of the theory formation session, and HR will work autonomously.

A final enhancement is that the user is now able to instruct HR to prove certain conjectures in a much more fine-grained manner. This extra functionality includes telling HR not to prove conjectures of a certain type, and telling it to use different axioms for conjectures of different types. This is enabled by the Java interpreter, so that, even at run-time, the user can specify fairly complicated conditions on conjectures. For example in experiment 2 below, we were only interested in conjectures about concepts of arity 1, so we told HR not to prove any conjectures of arity 2 or more. This meant that the session time reduced from around 5 hours to around 2 hours.

3.5. Enhancements to Counterexample Finding

We have enabled the user to step in and check whether certain objects of interest – which they supply – are counterexamples to a particular conjecture HR has made. Moreover, in number theory, if a user suspects that a counterexample may lie in a certain range, they can specify a lower and upper bound on a set of integers, and HR checks if any integer in the set breaks the conjecture. To perform the check, HR invokes Maple to calculate the user-given functions for each integer. Using this information, HR calculates examples of the concepts in the conjecture for each integer and tests whether the conjecture still holds. This functionality is useful once HR has identified the interesting conjectures in a session, as the user can choose one and test it empirically before attempting a proof (as we do in §4). Once either the user or HR has found a counterexample for a particular conjecture, the user can instruct HR to check whether any other implicates in the theory are broken by the same counterexample. This method was fairly effective in the second experiment below, as the introduction of only two counterexamples led to the breaking of 21 false conjectures.

3.6. Enhancements to Presenting Results

HR has a tendency to restate a theorem in many different ways. To reduce this, we have implemented a routine whereby HR can discard a conjecture if it follows directly from a previously proved theorem. Discarding such conjectures before trying to prove them not only increases the comprehensibility of the theory produced, but also increases efficiency, as invoking the theorem prover costs time. As an example, in experiment 2 below, HR makes (and Otter proves) the conjecture that:

$$\forall\ a, b\ (tau(a) = b \wedge isprime(a) \rightarrow iseven(b)).$$

Later, it makes this conjecture:

$$\forall\ a, b\ (tau(a) = b \wedge tau(b) = a \wedge isprime(b) \rightarrow iseven(a)).$$

However, rather than trying to prove the latter conjecture, HR discards it because it follows directly from the former conjecture.

To enable HR to discard such conjectures before trying to prove them, we implemented a subsumption checking algorithm. Given conjectures X and Y, this algorithm determines whether there is some unification of variables such that the goal of X unifies with the goal of Y, and the body of X unifies with a subset of the clauses in the body of Y. In such cases, it is easily shown that, if X is true, then Y is true. Whenever a new implicate is made, HR checks it against all the implicates currently in the theory, and discards it if it can be subsumed by a previous one. We have found that this greatly reduces the number of implicates presented to the user. However, as discussed in experiment 2 below, there is room for improvement in the efficiency of the subsumption checking algorithm.

Note also that, as in experiment 2, the user can instruct HR to discard any conjecture which can be subsumed by another *open* conjecture. This will reduce the number of conjectures produced, but there is an important caveat: if the subsuming conjecture turns out to be false, the discarded conjecture may not have been false, and may have been more interesting. As with any heuristic, however, certain sacrifices in completeness have to be made in order to increase the yield of interesting results.

HR's report generation is handled by the Java interpreter. Thus the reporting mechanism now has access to the entirety of HR's theory, which can be very advantageous, as it is not always possible to prescribe in advance what information will be required from a theory formed over several hours. For instance, the session described in experiment 2 below took just under 2 hours to complete. Unfortunately, the report scripts we had intended to use produced too many conjectures to view. However, as the scripts were interpreted, we were able to tweak them until they gave us exactly the output we were looking for.

4. Experiments and Results

In §5 we discuss a planned application of HR to discovery in pure mathematics, for which the interface with Maple will be very important. Our aims for this paper were (a) to show that the pruning measures discussed above are effective (b) to illustrate how we envisage HR being used in a research environment and (c) to demonstrate that it is possible to find interesting conjectures about Maple functions using HR and Otter as described above. In the first experiment below, we concentrate on the effectiveness of the pruning methods employed. In the second experiment, we return to a domain – refactorable numbers – which has been fruitful using alternative conjecture making strategies, to see if the current approach can re-discover previous results, and perhaps discover something new.

4.1. Experiment 1

For this experiment, HR was given as background knowledge three functions from the Maple `numtheory` package. The three functions were `tau(n)`, which

calculates the number of divisors of n, `sigma(n)`, which calculates the sum of divisors of n, and `isprime(n)`, which tests whether or not n is a prime number. We gave HR only the number 1 to start with, but gave it access to the numbers 2 to 30 from which to find counterexamples to false conjectures. Using a complexity limit of 6, we ran a breadth first search to completion using the compose, exists and split production rules. We also enabled applicability conjecture making, so that HR could make applicability conjectures when concepts applied to 2 or fewer objects of interest. This meant that the disjunct production rule was also used to produce concepts sporadically. We specified that HR should produce conjectures through equivalence checking, non-existence checking and subsumption checking. We also specified that it should extract implicates from these conjectures and that it should keep only the implicates. Finally, we specified that it should use Otter to try to prove any implicates produced. Otter's time limit was set to 5 seconds. After experimentation, we decided not to extract prime implicates, as this was computationally expensive and mostly fruitless in this domain.

The session took around 2 minutes on a Pentium 500Mhz processor, and lasted for 378 theory formation steps. HR produced 48 concepts. Due to the composition of functions, HR called Maple on 120 occasions, to calculate `isprime`, `tau`, and `sigma` for integers ranging from 1 to 195 (which is the sum of the divisors of 72). HR also introduced the numbers 2, 3, 4, 5, 6, 9 and 16 as counterexamples to false conjectures. These false conjectures were made in the following order, and are given with the counterexample HR found to disprove them:

```
all a b (((tau(a)=b) <-> (sigma(a)=b))) [counterexample = 2]
all a b (((tau(a)=b) <-> (tau(a)=b & tau(b)=a))) [3]
all a b (((sigma(a)=b) <-> (sigma(a)=b & tau(b)=a))) [4]
all a ((isprime(a)) <-> ((a=2 | a=3))) [5]
all a ((a=2 | a=4) <-> (isprime(sigma(a)))) [9]
all a b (tau(a)=b <-> tau(a)=b & tau(sigma(b))=b) [6]
all a b (tau(a)=b & isprime(b) -> tau(a)=b & tau(sigma(b))=b) [16]
```

In the session, HR produced 137 implicates. Of these, 43 had already been proved by Otter, including ones which followed from a calculation on particular integers, such as:

```
(68) all a ((a=2 | a=3) -> tau(sigma(a))=a)
```

Otter could prove this because HR gave it ground instances such as `tau(3)=2` and `sigma(2)=3`. There were also theorems which didn't follow from calculations, but were still obviously true, such as:

```
(56) all a b (tau(a)=b & sigma(b)=a & isprime(b) -> tau(sigma(b))=b)
```

Of the 94 conjectures which remained unsolved, we looked through the first 10 which were produced and added these 9 as axioms:

```
(0) all a (exists b (tau(a)=b))
(1) all a (tau(a)=1 -> a=1)
(3) all a (isprime(a) -> tau(a)=2)
(4) all a (tau(a)=2 -> isprime(a))
(5) all a (exists b (sigma(a)=b))
(7) all a (sigma(a)=1 -> a=1)
(8) all a b (tau(a)=b & sigma(a)=b -> tau(b)=a)
(9) all a b (tau(a)=b & sigma(a)=b -> sigma(b)=a)
(10) all a b (sigma(a)=b & sigma(b)=a -> tau(a)=b)
```

Conjectures (2) and (6) are missing from the above list because they were proved, hence not in the list of unsolved conjectures that HR presented to us. The conjecture we did not add from the first 10 unsolved ones was:

```
(11) all a b (tau(a)=b & isprime(a) -> isprime(b))
```

which we thought should follow from the other axioms, so we left it out. We see that HR has identified the definition of prime numbers in conjectures (3) and (4): `all a (isprime(a) <-> tau(a)=2)`. We also looked through the unsolved conjectures which were instantiations, and added these three as axioms:

```
(15) all a b (sigma(a)=b & sigma(b)=a -> a=1)
(21) all a (tau(tau(a))=a -> (a=1 | a=2))
(135) all a (a=3 -> isprime(sigma(sigma(a))))
```

Having given HR the additional axioms, we then asked it to attempt to re-prove all the unsolved conjectures. This was very effective, and reduced the number of unsolved conjectures from 94 to 22. We looked at the 17 unsolved conjectures which were not instantiations, and ordered these in terms of a measure of interestingness which was obtained by averaging the normalised applicability and normalised surprisingness. At the top of the ordered list was conjecture number 46, which we found very interesting:

```
(46) all a (isprime(sigma(a)) -> isprime(tau(a)))
```

Paraphrased, this states that, if you take an integer and add up the divisors, then if the result is a prime, the number of divisors you have just added up will also be prime. We used HR to check this conjecture empirically for the numbers 1 to 100, and it used Maple to perform the appropriate calculations. The empirical test was positive, so we tried to prove this conjecture, which we managed, as reported in appendix A. We then added conjecture 46 as an axiom and asked HR to attempt to prove the remaining unsolved conjectures in the light of this theorem. This reduced the unsolved non-instantiation conjectures to the following 10, ordered in terms of the interestingness measure mentioned above:

```
(127) all a (tau(tau(a))=a -> tau(sigma(sigma(a)))=sigma(a))
(129) all a (tau(tau(a))=a -> tau(sigma(a))=a)
```

```
(130) all a (tau(sigma(a))=a & tau(sigma(sigma(a)))=sigma(a) ->
      tau(tau(a))=a)
(64) all a b (sigma(a)=b & isprime(a) & isprime(b) -> tau(sigma(b))=b)
(111) all a b (sigma(a)=b & isprime(sigma(b)) -> isprime(tau(a)))
(90) all a b (sigma(a)=b & isprime(tau(b)) -> isprime(tau(a)))
(128) all a (tau(sigma(sigma(a)))=sigma(a) -> tau(sigma(tau(a)))=tau(a))
(108) all a b (sigma(a)=b & isprime(sigma(b))  -> tau(b)=a)
(47) all a b (sigma(a)=b & isprime(a) & isprime(b) -> tau(b)=a)
(109) all a b (sigma(a)=b & isprime(sigma(b)) -> isprime(a))
```

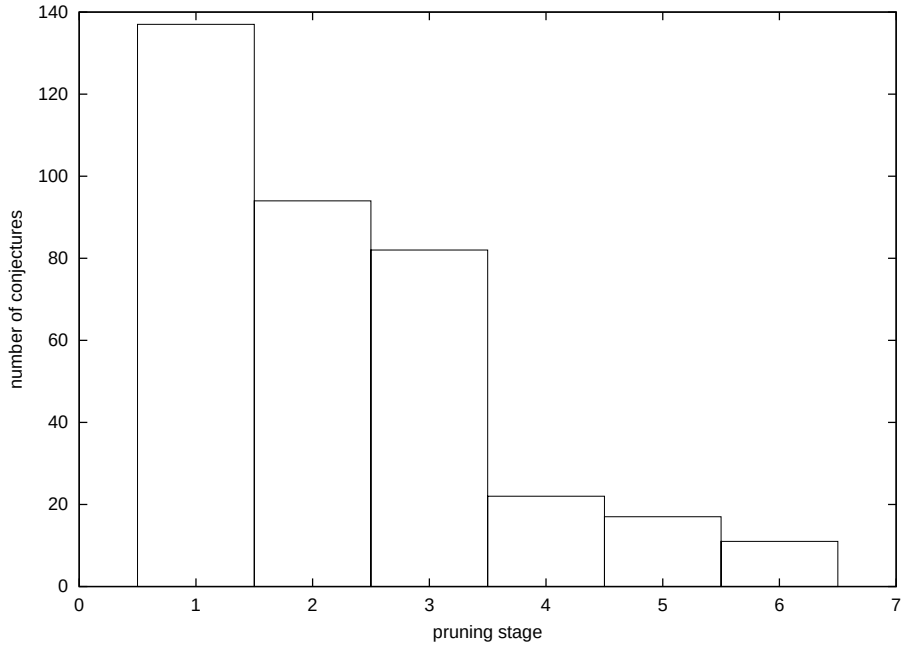

Figure 3: Pruning of conjectures in stages: stage 1 (all the conjectures), stage 2 (after using Otter to discard trivially true results), stage 3 (after the user chose conjectures to add as axioms), stage 4 (after another round of proving using the additional axioms), stage 5 (after pruning instantiation conjectures), stage 6 (after a final round of proving with a single new axiom added).

We note that conjectures (127) and (129) above should have been proved because we gave HR conjecture (21) as an axiom, which states that, given the left hand side of conjecture (127) or (129), then a = 1 or a = 2. However, we found that Otter could not prove either conjecture (with default settings), even when allowed five minutes to prove them. This is an anomaly we are currently investigating. We must also determine the significance – if any – of the other results. However, we feel it is a success that, in such a short session with HR, it managed to find a non-trivial conjecture of enough interest that a generalised theorem (see appendix A) was found and proved with some difficulty. Also, we

hope to have demonstrated that the pruning using Otter and the user to prove easy theorems worked well. In figure 3, we show the decrease in the number of unsolved conjectures at various stages of the session, and we note that the number of unsolved conjectures presented to the user was reduced from 137 to 11, a manageable number.

4.2. Experiment 2

To give some indication of how the combined HR/Otter/Maple system might be used in research, we look at a domain which we have previously explored. As described in [3], we used HR to discover novel, interesting integer sequences worthy of the Encyclopedia of Integer Sequences [26]. HR invented the concept of refactorable numbers, which are such that the number of divisors is itself a divisor, e.g., 9 is refactorable, because 9 has 3 divisors (1, 3 and 9) and 3 divides 9. This sequence was missing from the Encyclopedia, but had, in fact, been invented in 1990 [17]. Previously, we used an invent and investigate technique using the Encyclopedia itself to make conjectures about refactorable numbers, as described in [3] and [7].

In order for us to use the combined system described here to investigate refactorable numbers, we wrote a short Maple program which tested whether a given number was refactorable or not. We then gave HR the background concept of refactorable numbers and specified that it should use this file for checking whether an integer is refactorable. We also told HR to use the `tau`, `sigma` and `isprime` functions from Maple's `numtheory` package and gave it the concept of even and odd numbers, supplied with Java code for parity testing.

We ran HR for 10,000 steps and allowed it to produce concepts up to complexity 8. We knew in advance that the number of conjectures produced by HR would be substantially larger than in the previous experiment. For this reason, we decided to run some brief testing sessions, to determine some general results to give to Otter as axioms *before* our main theory formation session. We believed that giving Otter more general axioms to work with in advance would be a better approach in this situation than using the approach in experiment 1, where HR highlighted axioms *after* theory formation, (which were used in an attempt to prove and discard further theorems).

After a few short test sessions, where we added successively more axioms, we decided upon the following set to be given to Otter in the main session:

```
all a (tau(a) = 2 <-> isprime(a)).
all a (tau(a) = a <-> (a=1 | a=2)).
all a (tau(tau(a)) = a <-> (a=1 | a=2)).
all a (sigma(a) = a <-> a=1).
all a (isprime(a) -> (a=2 | isodd(a))).
all a (sigma(a) != 2).
all a (exists b (tau(a)=b)).
all a (exists b (sigma(a)=b)).
```

```
all a (tau(a)=1 <-> a=1).
all a (sigma(a)=1 <-> a=1).
all a (sigma(a)=a <-> a=1).
all a (sigma(sigma(a))=a <-> a=1).
all a (iseven(a) <-> -(isodd(a))).
all a (tau(a)=sigma(a) <-> a=1).
all a (issquare(a) <-> isodd(tau(a))).
```

Other than changing the complexity limit to 8, the setup was as for experiment 1, with one exception: we tailored the proof mechanism to ignore conjectures which were about concepts of arity greater than 1. By concentrating on conjectures of arity 1, we restricted our interest to those about number types. The session took 5883 seconds on a Pentium 2Mhz processor. HR produced 959 implicates about number types, but after pruning those proved by Otter, this number reduced to 184. We looked through the first ten conjectures produced, which were:

```
(20) all a (isprime(a) & isrefactorable(a) -> a=2)
(29) all a (isrefactorable(a) & isodd(a) -> issquare(a))
(30) all a (isrefactorable(a) & isodd(a)) -> (a=1 | a=9))
(65) all a (isprime(sigma(a)) -> isprime(tau(a)))
(84) all a (isrefactorable(a) & tau(a)=2 -> a=2)
(99) all a (isrefactorable(a) & issquare(a) & iseven(a) -> a=36)
(108) all a (issquare(a) -> isodd(sigma(a)))
(110) all a (isrefactorable(a) & isodd(a) -> isodd(sigma(a)))
(133) all a (isprime(a) & isodd(a) -> iseven(sigma(a)))
(134) all a (iseven(sigma(a)) -> iseven(tau(a)))
```

For each of these conjectures, we either found a counterexample to disprove it, or added it to the axioms as follows: we first noted that conjectures (20) and (29) were previously proved results. That is, in [3], we prove that 2 is the only prime refactorable number, and odd refactorable numbers are square numbers. We next noted that conjecture (30) was false, as there are an infinite number of odd refactorables: the next odd refactorable number after 9 is 225, and we added this as a counterexample. We then used HR to determine whether 225 was a counterexample to any other conjectures, and it falsified 18 more conjectures with this counterexample.

Conjecture (65) was the interesting result we found in experiment 1, and we proved conjecture (84), by showing that it follows from conjecture (20). We next showed that conjecture (99) was false, as there are an infinite number of even square refactorables, with the next being 3600. On adding this as a counterexample, HR reported that it broke one other false conjecture, which was not about refactorable numbers:

```
(781) all a (issquare(a) & iseven(a) -> isprime(tau(tau(a))))
```

Conjecture (108) was perhaps the first interesting result that HR found. This states that the sum of the divisors of a square number is odd. That the *number* of divisors of a square number is odd is a well known, simple, result, indeed, we gave this as an axiom to Otter. However, proving conjecture (108) was not as straightforward. The proof is given in appendix B.

The proof of conjecture (110) followed from those of conjectures (29) and (108). This is a new result about refactorable numbers: the sum of divisors of an odd refactorable is itself odd. We easily proved conjecture (133): the sum of divisors of an odd prime is even because it is 1 plus the prime. We found conjecture (134) more interesting: if an integer has an even sum of divisors, then it has an even number of divisors. This also follows from the truth of conjecture (108), and is similar in nature (and perhaps interestingness) to the conjecture from experiment 1 (that if an integer has a prime sum of divisors, it will have a prime number of divisors).

Using the counterexamples, we removed 21 conjectures, and then we added conjectures (20), (29), (65), (84), (108), (110), (133) and (134) as axioms, and asked HR to attempt to prove those still remaining. To save time, before re-proving, we removed the conjectures which were instantiations, such as conjectures (20) and (30) above, as we decided that these were less interesting than the others. This reduced the number of open conjectures to 112. After removing all theorems proved in the re-proving session, we were left with 69 open conjectures. In this round of proving, Otter managed to prove some results which could have distracted our attention, because they appear interesting. For instance, Otter proved these theorems:

```
(220) all a (iseven(a) & isprime(tau(a)) -> isodd(sigma(a)))
(301) all a (isprime(a) & isrefactorable(sigma(a)) -> iseven(sigma(a)))
(414) all a (isrefactorable(a) & iseven(sigma(a)) -> iseven(a))
```

As a final pruning measure, we removed any conjectures with applicability less than 0.1. This left us with 26 open conjectures, which we present in appendix C. We have not yet fully investigated these remaining conjectures, and it seems likely that the majority may be false. Of particular interest to us are the conjectures about refactorable numbers: amongst others, HR made the conjectures that: (i) for even numbers, if $\sigma(a)$ is refactorable, then $\tau(a)$ and $\sigma(a)$ will be even (ii) for odd numbers, if $\sigma(a)$ is even and refactorable, then $\tau(\tau(a))$ and $\sigma(\tau(a))$ will both be prime (iii) if $\tau(a)$ is refactorable and $\tau(\tau(a))$ is prime, then $\sigma(\tau(a))$ will also be prime, and (iv) if both $\sigma(a)$ and $\sigma(\sigma(a))$ are refactorable, then $\tau(\sigma(a))$ will be refactorable and $\sigma(\tau(a))$ will be odd.

In this experiment, the time spent by HR checking whether one implicate subsumed another was twice as long as the time spent using Otter to prove theorems, which is normally the most time consuming exercise. Hence, there is room for improvement in the efficiency of the subsumption checking. We plan to employ a Prolog implementation to perform this task.

5. Conclusions and Further Work

In the same way that pure mathematicians do not in general need a machine to prove theorems for them, they similarly have little need for conjectures to be produced automatically. However, the popularity of computer algebra systems is undeniable. These systems help mathematicians to both prove/disprove existing hypotheses and to generate new ideas and discover new hypotheses. We seek to enhance the latter phenomenom by automating some of the processes which occur between a mathematician specifying a function to a computer algebra system and making a discovery about that function. In the final sentence of [5], we state that, if automated conjecture making such as that employed by HR

> "... can be embedded into computer algebra systems, we believe that theory formation programs will one day be important tools for mathematicians." (page 301)

The work presented here represents the first step towards using automated theory formation to enable computer algebra systems to intelligently make conjectures about functions the user is experimenting with. This complements our work on data-mining the Encyclopedia of Integer Sequences to find conjectures [3, 7], which also led to discoveries in number theory. Moreover, we have recently combined HR, Otter and Maple into an integrated system called Homer, as described in [9].

Other approaches to making research conjectures for mathematicians have either performed an exhaustive search for theorems using the power of an efficient theorem prover, or have required bespoke programs. For instance, in [21], McCune uses an exhaustive search with Otter to find interesting new axiomatisations of group theory and other algebras. Similarly, in [2], Chou used the power of Wu's method to find new constructions in plane geometry. The Graffiti program [13] has produced scores of conjectures which the graph theory community have proved and disproved, but this is a graph theory specific program which isn't publicly available. To our knowledge, HR is the only program which uses both computer algebra and theorem proving systems to make research conjectures.

We have shown how HR can be used to make conjectures about Maple functions chosen by the user. Given HR's current abilities to form concepts and make conjectures, the main technical difficulty to overcome for this project was to reduce the number of uninteresting conjectures produced. To do this, we used much of HR's functionality, including:

[1] Its ability to call Otter to prove theorems from first principles. Such theorems are likely to be uninteresting, and hence can be discarded. In experiment 1, this enabled HR to discard 31% of the 137 implicate conjectures HR produced in total, and in experiment 2, this figure rose to over 80%. For this to be effective in number theory, we enabled HR to pass calculations from Maple to Otter.

[2] A new ability, which allows the user to choose some of HR's conjectures to add as axioms. Subsequent attempts to prove the unsolved conjectures allows more

pruning of the theorems because they can be proved from first principles and the (usually simple) axioms added by the user. In experiment 1, after giving some of HR's obviously true theorems to Otter as axioms, this reduced the number of unsolved conjectures from 94 to 22, an acceptable number for the user to look through.

[3] The ability to extract simply stated implicates and order conjectures in terms of measures of interestingness, so that the user can browse the most interesting conjectures first.

The second point above represents a first step towards a more interactive environment for theory development within HR. We hope to pursue such an interactive mode – similar to that employed by Lenat with his AM program [18] – by allowing the user to step in and provide new concepts, conjectures, theorems, proofs and counterexamples at will during the theory formation session. This will be useful for an extended application to mathematical discovery planned for HR: the exploration of the domain of Zariski spaces developed by McCasland et al. [19]. Due to the relative complexity of this domain, an interactive mode in HR will be essential. Also, HR's links via MathWeb to various pieces of mathematical software including provers such as Otter, Spass and E, model generators such as MACE, computer algebra systems such as Maple and Gap, and constraint solvers such as Solver, will be essential for this project. Our aim for the HR system is for the theory behind it to encompass more and more abilities, while the tasks reliant on HR's code become fewer, as HR interfaces with more mathematics programs.

The application of HR to finding conjectures about CAS functions is still in its early stages. Our choice of which Maple functions to form conjectures about was inspired by working with these functions in a different project [7], but in general, the user will specify a much larger set of functions. HR must therefore decide which ones to use, possibly discarding some after an initial investigation reveals that there are very few interesting properties about which it can make conjectures. Furthermore, we need to undertake extended testing of HR to highlight its strengths and limitations when working with CAS functions. To this end, we have been working with the mathematician Sophie Huczynska, who used HR (embedded in the Homer system [9]), to produce research conjectures in number theory. Our initial testing has been encouraging: in a four hour* session, Dr. Huczynska found four "number theoretically interesting" conjectures about the ϕ function (which counts the number of integers less than and co-prime to an integer). For a report on this endeavour, including proofs that (a) $\forall\ n > 2, \phi(n)$ is even and (b) $\phi(n)$ is square implies that $\tau(n)$ is even (two theorems found by Homer and proved by Dr. Huczynska), see [9].

Finally, we need to improve the integration of HR and Maple, in terms of the dialogue between them and the way in which that communication is performed. In future, we envisage a more sophisticated interface between Maple and HR,

*HR was running for approximately 15 minutes of the four hours.

in particular, enabling HR to write conjectures in a format Maple can read, then using Maple to check them empirically (over a large set of integers, or graphs, etc.). This way of interacting would improve the efficiency of checking the conjectures, as HR is not as optimised as Maple for performing lengthy calculations. We also plan to enable HR to talk to Otter and Maple via the MathWeb software bus. We hope to have shown here the potential for using HR to discover interesting facts about computer algebra functions and concepts related to them. As the most popular pieces of software within pure mathematics are computer algebra systems, it is essential that HR is able to interact with them, and it is a long-term goal of ours to embed HR's discovery functionality into computer algebra systems such as Maple.

Acknowledgements

This work has been supported in part by EPSRC grant GR/M98012. This research was inspired by discussions with Jacques Calmet and Clemens Ballarin during the author's visit to Karlsruhe University, funded by the European Union IHP grant CALCULEMUS HPRN-CT-2000-00102. As with all projects involving HR, the input from Alan Bundy and Toby Walsh has been essential. We would also like to thank the anonymous referees from the Calculemus conference for their interesting comments and suggestions about this work, and the organisers of that conference, at which this material was first presented. We would also like to thank the referees of the journal paper for further interesting and important comments. Finally, we wish to thank Ravi Jain, Noor Mohamadali, Roy McCasland and Sophie Huczynska for using HR in mathematical domains, as their input has led to much improved implementations of HR.

References

[1] B Buchberger. Theory exploration versus theorem proving. In *Proceedings of Calculemus 99, Systems for Integrated Computation and Deduction*, 1998.

[2] S Chou. Proving and discovering geometry theorems using Wu's method. Technical Report 49, Computing Science, University of Austin at Texas, 1985.

[3] S Colton. Refactorable numbers - a machine invention. *Journal of Integer Sequences*, 2, 1999.

[4] S Colton. Automated theory formation applied to mutagenesis data. In *Proceedings of the 1st British-Cuban Workshop on BioInformatics*, 2002.

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

[6] S Colton, A Bundy, and T Walsh. Automatic identification of mathematical concepts. In *Machine Learning: Proceedings of the 17th International Conference*, 2000.

[7] S Colton, A Bundy, and T Walsh. Automatic invention of integer sequences. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, 2000.

[8] S Colton, A Bundy, and T Walsh. On the notion of interestingness in automated mathematical discovery. *International Journal of Human Computer Studies*, 53(3):351–375, 2000.

[9] S Colton and S Huczynska. The Homer system. In *Proceedings of the 19th International Conference on Automated Deduction (LNAI 2741)*, pages 289–294, 2003.

[10] S Colton and I Miguel. Constraint generation via automated theory formation. In *Proceedings of CP-01*, 2001.

[11] L De Raedt and L Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.

[12] L Dehaspe and H Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.

[13] S Fajtlowicz. On conjectures of Graffiti. *Discrete Mathematics 72*, 23:113–118, 1988.

[14] Andreas Franke and Michael Kohlhase. System description: MathWeb, an agent-based communication layer for distributed automated theorem proving. In *Proceedings of CADE-16*, pages 217–221, 1999.

[15] Gap. *GAP Reference Manual*. The GAP Group, School of Mathematical and Computational Sciences, University of St. Andrews, 2000.

[16] G Hardy and E Wright. *The Theory of Numbers*. Oxford University Press, 1938.

[17] R Kennedy and C Cooper. Tau numbers, natural density and Hardy and Wright's theorem 437. *International Journal of Mathematics and Mathematical Sciences*, 13:383–386, 1990.

[18] D Lenat. AM: Discovery in mathematics as heuristic search. In D Lenat and R Davis, editors, *Knowledge-Based Systems in Artificial Intelligence*. McGraw-Hill Advanced Computer Science Series, 1982.

[19] R McCasland, M Moore, and P Smith. An introduction to Zariski spaces over Zariski topologies. *Rocky Mountain Journal of Mathematics*, 28:1357–1369, 1998.

[20] W McCune. The OTTER user's guide. Technical Report ANL/90/9, Argonne National Laboratories, 1990.

[21] W McCune. Single axioms for groups and Abelian groups with various operations. *Journal of Automated Reasoning*, 10(1):1–13, 1993.

[22] W McCune. A Davis-Putnam program and its application to finite first-order model search. Technical Report ANL/MCS-TM-194, Argonne National Laboratories, 1994.

[23] W McCune. Solution of the Robbins problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.

[24] S Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

[25] D. Redfern. *The Maple Handbook: Maple V Release 5.* Springer Verlag, 1999.

[26] N Sloane. The Online Encyclopedia of Integer Sequences. *http://www.research.att.com/~njas /sequences*, 2000.

[27] J Zimmer, A Franke, S Colton, and G Sutcliffe. Integrating HR and tptp2x into MathWeb to compare automated theorem provers. Technical report, Division of Informatics, University of Edinburgh, 2001.

A. Proof that $isprime(sigma(n)) \rightarrow isprime(tau(n))$

Lemma
For all n, $\tau(n)$ is prime $\iff$ $n = p^{q-1}$ for primes p and q.

Proof
If $n = p^{q-1}$ then $\tau(n) = q$, hence $\tau(n)$ is prime. Conversely, suppose that the prime factorisation of n is $p_1^{k_1} \dots p_l^{k_l}$, and that $\tau(n)$ is prime. Now $\tau(n) = (k_1 + 1) \dots (k_l + 1)$, hence $l = 1$, and n must be of the form p^a for some a. So, $\tau(p^a) = a + 1$, and a must be one less than a prime, q.

Lemma 2
If the prime factorisation of integer n is: $n = \prod_{i=1}^{l} p_i^{k_i}$, then

$$\sigma_m(n) = \prod_{i=1}^{l} \left(\frac{p_i^{m(k_i+1)} - 1}{p_i - 1} \right).$$

(Where $\sigma_m(n)$ is the sum of the mth powers of the divisors of n). For the proof of this result, see theorem 274 of [16]. We also need the following well known identity:

$$\frac{a^b - 1}{a - 1} = 1 + a^2 + \ldots + a^{b-1} = \sum_{i=0}^{b-1} a^i.$$

Theorem
$\forall\ m, n \in \mathbf{N}, \quad \tau(\sigma_m(n)) = 2 \Rightarrow \tau(\tau(n)) = 2.$

Proof
Let the prime factorisation of n be $p_1^{k_1} \ldots p_l^{k_l}$, and let m be an integer. Suppose also that $\tau(\sigma_m(n)) = 2$, i.e. that $\sigma_m(n)$ is prime. We see from lemma 2 that $\sigma_m(n)$ has at least $l+1$ factors (counting 1 as well). Therefore, as $\sigma_m(n)$ is prime, $l = 1$. Hence we can write $n = p^a$ for some prime p and some $a \in \mathbf{N}$. If we assume that $\tau(n)$ is composite, then $\tau(n) = a + 1 = xy$ for some $x, y \in \mathbf{N}, x > 1, y > 1$. Hence $a = xy - 1$. So, using lemma 2 again:

$$\begin{aligned}
\sigma_m(n) &= \frac{p^{m(a+1)} - 1}{p - 1} = \frac{p^{m(xy-1+1)} - 1}{p - 1} = \frac{p^{mxy} - 1}{p - 1} \\
&= \frac{(p^{mx} - 1)(p^{(y-1)mx} + p^{(y-2)mx} + \ldots + p^{mx} + 1)}{p - 1} \\
&= \frac{p^{mx} - 1}{p - 1} \sum_{i=1}^{y} p^{(y-i)mx} \;=\; \left(\sum_{i=0}^{mx-1} p^i \right) \cdot \left(\sum_{j=1}^{y} p^{(y-j)mx} \right)
\end{aligned}$$

As $x > 1$ and $y > 1$, neither of the factors in this final product equal 1. Hence, this provides a contradiction, because $\sigma_m(n)$ is prime. Hence our assumption that $\tau(n)$ is composite must be false, and we see that $\tau(n)$ is a prime. □

Corollary
Taking $m = 1$ above, we see that: $\forall\ n \in \mathbf{N}, \quad \tau(\sigma(n)) = 2 \Rightarrow \tau(\tau(n)) = 2$, i.e, if the sum of divisors of n is prime, then the number of divisors of n will be prime.

B. Proof that $issquare(a) \rightarrow odd(\sigma(a))$

• Suppose n is an odd square number. Each divisor of n must be odd, and, because n is a square, we know that $\tau(n)$ is odd. $\sigma(n)$ is therefore the sum of an odd number of odd numbers, so must be an odd number itself.

• Suppose now that n is an even square number. Then $n = 2^{2x} {p_1}^{k_1} \ldots {p_m}^{k_m}$ for some number $x > 0$ and some odd primes p_i. The number of even divisors of n will therefore be: $2x\,(\sum_{i=1}^{m}(k_i + 1))$. Hence, n must have an even number of even divisors. Again, we know that, because n is square, it has an odd number of divisors in total, so it must have an odd number of odd divisors. $\sigma(n)$ is therefore the sum of an even number of even numbers and an odd number of odd numbers. $\sigma(n)$ is therefore odd. □

C. Pruned Results from Experiment 2

In the following conjectures, $even(a)$ means that a is even, $odd(a)$ means that a is odd, $ref(a)$ means that a is refactorable, $prime(a)$ means that a is prime and $square(a)$ means that a is a square number. $\sigma(a)$ is the sum of the divisors of a, and $\tau(a)$ is the number of divisors of a. The integers in square brackets are those which satisfy the definition on the left hand side of the implicate.

$even(a), ref(\sigma(a)) \rightarrow even(\tau(a))$ [6, 10, 14, 22, 24, 28, 30, 38, 42, 44, 46]

$even(a), ref(\sigma(a)) \rightarrow even(\sigma(a))$ [6, 10, 14, 22, 24, 28, 30, 38, 42, 44, 46]

$odd(a), even(\tau(a)) \rightarrow even(\sigma(a))$
[3, 5, 7, 11, 13, 15, 17, 19, 21, 23, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47]

$prime(a) \rightarrow \tau(\sigma(\tau(a))) = \tau(a)$ [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]

$even(a), square(\tau(a)) \rightarrow prime(\tau(\tau(a)))$ [6, 8, 10, 14, 22, 26, 34, 36, 38, 46]

$square(\tau(a)), even(\tau(a)) \rightarrow prime(\tau(\tau(a)))$
[6, 8, 10, 14, 15, 21, 22, 26, 27, 33, 34, 35, 38, 39, 46]

$odd(a), ref(\sigma(a)), even(\sigma(a)) \rightarrow prime(\tau(\tau(a)))$ [7, 11, 15, 17, 23, 27, 39]

$prime(\tau(\sigma(a))) \rightarrow prime(\tau(a))$ [2, 3, 4, 9, 16, 25]

$prime(\tau(\sigma(a))) \rightarrow prime(\tau(\tau(a)))$ [2, 3, 4, 9, 16, 25]

$even(a), square(\tau(a)) \rightarrow prime(\sigma(\tau(a)))$ [6, 8, 10, 14, 22, 26, 34, 36, 38, 46]

$square(\tau(a)), even(\tau(a)) \rightarrow prime(\sigma(\tau(a)))$
[6, 8, 10, 14, 15, 21, 22, 26, 27, 33, 34, 35, 38, 39, 46]

$odd(a), ref(\sigma(a)), even(\sigma(a)) \rightarrow prime(\sigma(\tau(a)))$ [7, 11, 15, 17, 23, 27, 39]

$ref(\tau(a)), prime(\tau(\tau(a))) \rightarrow prime(\sigma(\tau(a)))$
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 36, 37, 41, 43, 47]

$square(\tau(a)), prime(\tau(\tau(a))) \rightarrow prime(\sigma(\tau(a)))$
[6, 8, 10, 14, 15, 21, 22, 26, 27, 33, 34, 35, 36, 38, 39, 46]

$even(\tau(a)), prime(\tau(\tau(a))) \rightarrow prime(\sigma(\tau(a)))$ [2, 3, 5, 6, 7, 8, 10, 11, 13, 14, 15, 17, 19, 21, 22, 23, 26, 27, 29, 31, 33, 34, 35, 37, 38, 39, 41, 43, 46, 47]

$square(a), \tau(\sigma(\tau(a))) = \tau(a) \rightarrow ref(\tau(\tau(a)))$ [1, 4, 9, 25, 49]

$prime(\tau(\sigma(a))) \rightarrow ref(\tau(\tau(a)))$ [2, 3, 4, 9, 16, 25]

$even(a), ref(\tau(a)), even(\tau(a)) \rightarrow ref(\tau(\sigma(a)))$ [2, 24, 30, 40, 42]

$odd(a), ref(\tau(a)), even(\tau(a)) \rightarrow prime(a)$
[3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]

$ref(\sigma(a)), ref(\sigma(\sigma(a))) \rightarrow ref(\tau(\sigma(a)))$ [1, 14, 15, 23, 42]

$ref(\tau(a)), square(\tau(\tau(a))) \rightarrow ref(\tau(\sigma(a)))$ [1, 24, 30, 40, 42]

$even(a), square(\tau(\tau(a))) \rightarrow even(\tau(a))$ [12, 18, 20, 24, 28, 30, 32, 40, 42, 44, 48, 50]

$ref(\tau(a)) \rightarrow odd(\sigma(\tau(a)))$
[1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 24, 29, 30, 31, 36, 37, 40, 41, 42, 43, 47]

$odd(a), ref(\sigma(a)) \rightarrow odd(\sigma(\tau(a)))$ [1, 7, 11, 15, 17, 23, 27, 39]

$ref(\sigma(a)), ref(\sigma(\sigma(a))) \rightarrow odd(\sigma(\tau(a)))$ [1, 14, 15, 23, 42]

$odd(\sigma(\sigma(a))) \rightarrow odd(\sigma(\tau(a))$ [1, 3, 7, 10, 17, 21, 22, 30, 31, 46]