Making Conjectures about Maple Functions

Simon Colton

Division of Informatics
University of Edinburgh
United Kingdom
simonco@dai.ed.ac.uk
http://www.dai.ed.ac.uk/~simonco

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 by using the HR theory formation system to make conjectures about Maple functions supplied by the user. Experience has shown that HR produces too many conjectures which are 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 true but not trivially provable. By providing an application of HR's theory formation in number theory, we show that using Otter to prune HR's dull conjectures has much potential for producing interesting conjectures about standard computer algebra functions.

1 Introduction

There is an unfortunate dichotomy between the application of computer algebra systems (CASs) and automated theorem provers (ATPs) to pure mathematics: the concepts usually dealt with by computer algebra techniques are of too high a complexity to prove things about (at the moment) using automated techniques. 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 less 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 [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 the automated theorem provers and is unlikely to happen in the short term. If the aim of the integration of mathematical systems is to generate *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 provably true 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, as sophisticated conjecture making involves not only making conjectures about the given functions, but also about closely related functions. Hence, we will also be automating the first step and closing a cycle of theory formation.

We use the HR theory formation system [4] to produce conjectures about a set of computer algebra functions provided by the user. In particular, we will use functions from the Maple CAS [16] and we will use the Otter theorem prover [13] to prove some of HR's conjectures, so that we can discard them. In §2, we describe the core functionality behind HR which enables it to make conjectures. In §3 we describe the additional functionality implemented for this application to the generation of conjectures about Maple functions. In §4, we describe a session using HR to generate conjectures about some Maple functions from number theory, and in appendix A, we prove one of the results which HR discovered during the session.

2 Conjecture Making in HR

Much of HR's functionality was used for this application to making conjectures about Maple functions. Each functionality can be broadly characterised as part of one of five tasks: (i) inventing concepts (ii) making conjectures (iii) finding counterexamples (iv) proving theorems and (v) reporting results.

2.1 Inventing Concepts

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 session described in this paper are integers, and the background information is supplied in the form of Maple functions. From the background information, HR uses ten production rules to produce a new concept from one (or two) old concepts. The production rules are described in more detail in [5], and we concentrate here on four:

- The *compose* production rule composes functions using conjugation
- 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 `tau(n)` Maple function, which calculates the number of divisors of `n`. Using the compose production rule, HR invents the concept of pairs of integers, $[a, b]$ for which $b = tau(a)$ 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., $[a] : \exists\ b\ (tau(a) = b\ \&\ isprime(b))$. Hence HR has invented the concept of integers which have a prime number of divisors, a concept which we discuss further 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 [5]. Hence, the complexity of the concept in figure 1 is the number of boxes, i.e., four.

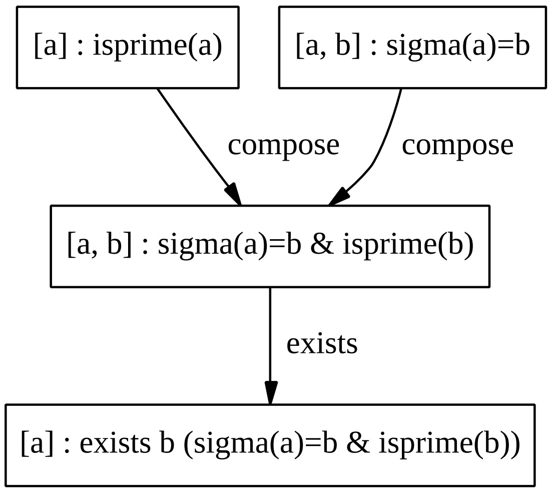
[a] : isprime(a)
[a, b] : sigma(a)=b
compose
compose
[a, b] : sigma(a)=b & isprime(b)
exists
[a] : exists b (sigma(a)=b & isprime(b))

Fig. 1. Construction of a number theory concept

2.2 Making Conjectures

HR has a number of ways to make conjectures, both by noticing empirical patterns and by extracting conjectures from others. 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, the concept is new and is added to the theory. When added to the theory, HR determines which concepts the new concept *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 two implicate conjectures: $f(a)$ & $g(a) \rightarrow h(a)$ and $f(a)$ & $g(a) \rightarrow x(a)$.

HR also extracts implicate conjectures from equivalence conjectures and non-existence conjectures. For instance if HR made the equivalence conjecture that $f(a)$ & $g(a) \leftrightarrow h(a)$ & $x(a)$, it would extract four implicates from this:

$$\begin{gathered} f(a) \ \& \ g(a) \rightarrow h(a) \\ f(a) \ \& \ g(a) \rightarrow x(a) \\ h(a) \ \& \ x(a) \rightarrow f(a) \\ h(a) \ \& \ x(a) \rightarrow g(a) \end{gathered}$$

Similarly, 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. Note that 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 tries the two prime implicates:

$$\begin{gathered} f(a) \rightarrow h(a) \\ g(a) \rightarrow h(a) \end{gathered}$$

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. To summarise, HR makes non-existence, equivalence and subsumption conjectures empirically, then extracts implicates from these and prime implicates, where possible, from the implicates.

2.3 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, thus whenever HR invents a concept, it will have less work to do to calculate the example set for the concept. Taking this to the extreme, in §4, we give HR only the number 1 to start with, but allow it access to the numbers 2 to 30 in order to find counterexamples to false conjectures. This not only increases efficiency, but 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 [15] to find counterexamples, but discussion of this is beyond the scope of this paper.

2.4 Proving Theorems

HR has some built-in abilities to decide when a conjecture it makes is trivially true, e.g., it can tell that the conjecture $f(a)$ & $g(a) \leftrightarrow g(a)$ & $f(a)$ is true. It also keeps a record of which concepts it generates are functions, so that it can tell that the conjecture $\nexists a(f(a) = k_1$ & $f(a) = k_2)$ is true, where k_1 and k_2 are different ground instances. In fact, it uses its primitive theorem proving to avoid inventing concepts such as this, because it knows in advance that the concept will have no examples, leading to a dull non-existence conjecture. If HR had any 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 via MathWeb [9, 17], but the application here was undertaken using a simple file interface.

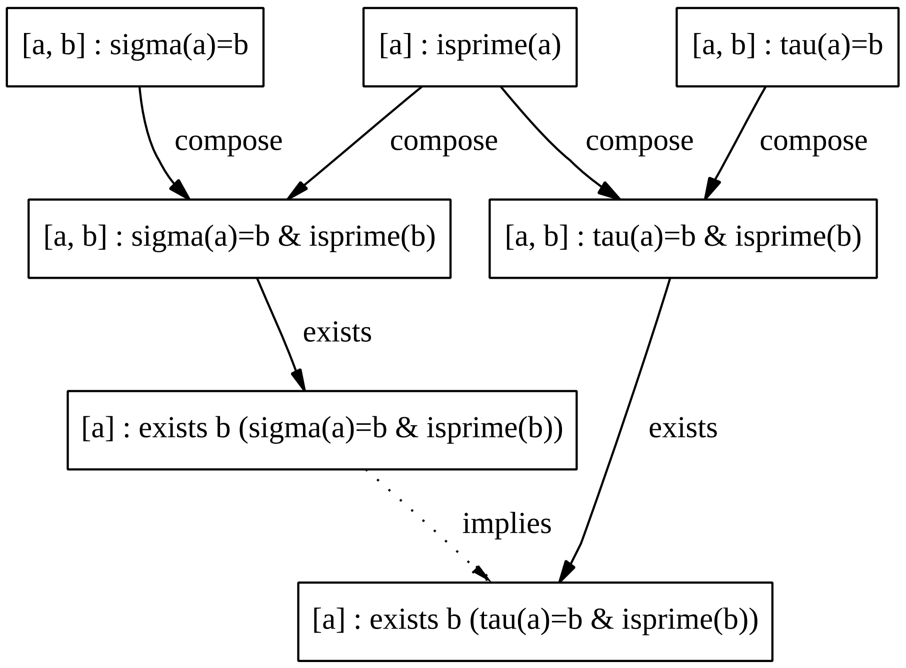
[a, b] : sigma(a)=b
[a] : isprime(a)
[a, b] : tau(a)=b
compose
compose
compose
compose
[a, b] : sigma(a)=b & isprime(b)
[a, b] : tau(a)=b & isprime(b)
exists
[a] : exists b (sigma(a)=b & isprime(b))
exists
implies
[a] : exists b (tau(a)=b & isprime(b))

Fig. 2. Construction of an implicate conjecture

2.5 Reporting Results

HR is able to prune the conjectures it produces and order those remaining in terms of measures of interestingness. In particular, for this application, we use HR to keep only implicates extracted from equivalence, non-existence and subsumption conjectures, as these are usually easier to understand. We also instruct HR to discard any conjectures which Otter can prove, as these are likely to follow from the definitions of the Maple functions provided and be fairly uninteresting.

Of those implicates remaining, we use two measures of interestingness to order them. Firstly, each implicate comprises a concept which implies a single clause. The *applicability* of a concept gives an indication of the scope of the conjecture, where the applicability 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 LHS of the conjecture. For instance, if HR had the integers 1 to 30 as objects of interest, then the concept of prime numbers would score 10/30 for applicability, as there are 10 prime numbers between 1 and 30, namely 2, 3, 5, 7, 11, 13, 17, 19, 23 and 29. Hence implicate conjectures where the concept which makes up the premises is the concept of prime numbers will score 1/3 for applicability. Conjectures with very low applicability tend to be uninteresting, so sorting the conjectures in terms of decreasing applicability can be useful.

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 just one concept. If two concepts conjectured to be related share many concepts in their construction paths, their definitions are likely to be similar, and the relationship between them will be less surprising, so they score less 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 too can be measured in terms of surprisingness.

3 Additional Functionality

Each new application of HR necessitates some new functionality. In this case, we have extended HR's functionality in all the five areas discussed in §2 above. In terms of improved concept formation, HR is now able to communicate with Maple. At present, it does this in the same way as it does with Otter, by reading a file, invoking Maple in such a way that it outputs answers to a file, and then reading that file. HR, Maple and Otter are already part of the MathWeb software bus [9] and we have been successful in enabling HR to invoke Otter (and other provers) via MathWeb [17]. 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.

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`. HR also calls Maple during concept formation, 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`. In future, we envisage a more sophisticated interface between Maple and HR, 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 interface would improve the efficiency of checking the conjectures, as HR is not optimised like Maple for performing lengthy calculations.

We also improved the way in which HR writes definitions, so that the conjectures about the concepts would be 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 was given the ability to collate and remove 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 number of divisors described in §2.1 above in this way:

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

which is easier to understand. This functionality is also useful for an application to constraint generation [7].

In terms of improved conjecture making and reporting, HR can now 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 being the number 2 and (c) the concept of 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 the 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.

In terms of improved theorem proving, we gave HR the ability to pass Otter ground instances of the Maple functions. For example, in §4, we describe a session with HR using the Maple `tau(n)` function, which counts the number of divisors of `n`. Because during that session, HR makes instantiations, it will eventually discover 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` and `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.

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 §4 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 also means that, to a certain extent, the user does not have to worry about specifying the axioms of the domain in advance, as HR will come across (some of) them. In fact, for the application in §4, we gave HR no axioms of number theory in advance.

In terms of improved counterexample finding, we enabled the user to step in and check whether certain objects of interest are counterexamples to a particular conjecture HR has made. In number theory, the user 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 Maple 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).

4 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 to show that the pruning measures discussed above are effective and that it is possible to find interesting conjectures about CAS functions using HR.

We give details here of a session with HR in number theory, where 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. 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 it produced. After experimentation, we decided not to extract prime implicates, as this was computationally expensive and mostly fruitless.

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 this order, 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 implicate conjectures. Of these, 43 had already been proven 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)))
```

Note that conjectures (2) and (6) were proved, hence not in the list of those 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 might 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 just 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 a number and add up the divisors, with the result being a prime number, then the coefficient of divisors you have just added up will also be a 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 just 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))
```

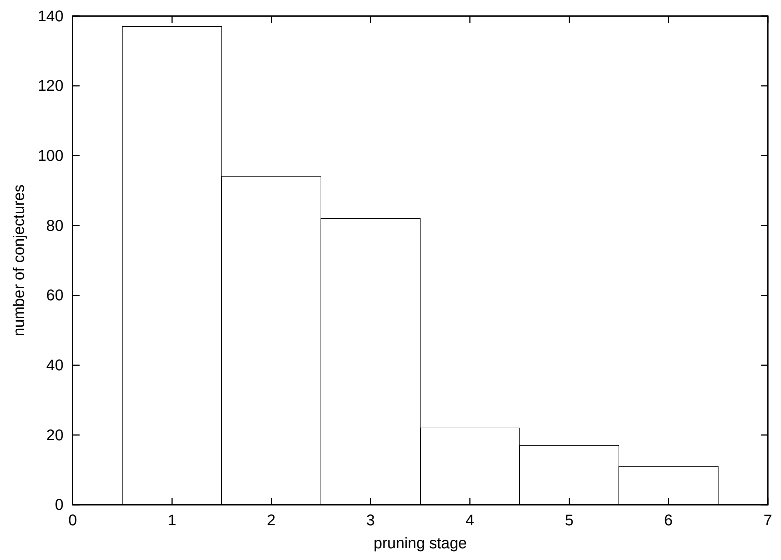

Fig. 3. Pruning of conjectures in stages: stage 1 (all the conjectures), stage 2 (after using Otter to discard trivially true results), stage 3 (after the user chose conjectures to add as axioms), stage 4 (after another round of proving using the additional axioms), stage 5 (after pruning instantiation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

We note that conjectures (127) and (129) above should be 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 think 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 just 11.

5 Conclusions and Further Work

In the final sentence of [4], we state that:

> "... if this technology can be embedded into computer algebra systems, we believe that theory formation programs will one day be important tools for mathematicians." (page 361)

The work presented here represents the first step towards using automated theory formation to enable computer algebra systems to intelligently make research conjectures about functions the user is experimenting with. It compliments, but is distinct to, our work datamining the Encyclopedia of Integer Sequences to find conjectures [3, 6] which also led to discoveries in number theory.

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 [14], McCune uses an exhaustive search with Otter to find 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 [8] has produced scores of conjectures which the graph theory community have eagerly 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 the HR theory formation system 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 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 our example session, this enabled HR to discard 43 such theorems (31% of the 137 implicate

conjectures HR produced in total). For this to be effective in number theory, we enabled HR to pass ground instances 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. After giving some of HR's obviously true theorems to Otter as axioms, this reduced the number of unsolved conjectures from 94 to just 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 [11] – 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 we have planned for HR: the exploration of the domain of Zariski spaces developed by Roy McCasland [12]. Due to the relative complexity of this domain, the 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 [6], 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 HR can make conjectures. Furthermore, we need to undertake extended testing of HR to highlight its strengths and limitations when working with CAS functions. Finally, we need to improve the integration of HR and Maple, in terms of (i) the communication between them (i.e., enable HR to write Maple functions so that Maple, rather than HR, can be used to check conjectures empirically) and (ii) the way in which that communication is performed (i.e., by enabling HR to talk to Otter and Maple via the MathWeb software bus).

We hope to have shown here a glimpse of 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 such programs, and it is a long-term goal of the HR project to embed HR's discovery functionality into computer algebra systems.

Acknowledgments

This work is supported by EPSRC grant GR/M98012. The author is also affiliated to the Department of Computer Science at the University of York. 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 for their interesting comments and suggestions about this work.

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 in Pure Mathematics.* PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 2000.
5. S Colton, A Bundy, and T Walsh. Automatic identification of mathematical concepts. In *Machine Learning: Proceedings of the 17th International Conference*, 2000.
6. S Colton, A Bundy, and T Walsh. Automatic invention of integer sequences. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, 2000.
7. S Colton and I Miguel. Constraint generation via automated theory formation. In *Proceedings of CP-01*, 2001.
8. S Fajtlowicz. On conjectures of Graffiti. *Discrete Mathematics 72*, 23:113–118, 1988.
9. 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.
10. G Hardy and E Wright. *The Theory of Numbers.* Oxford University Press, 1938.
11. 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.
12. R McCasland, M Moore, and P Smith. An introduction to Zariski spaces over Zariski topologies. *Rocky Mountain Journal of Mathematics*, 28:1357–1369, 1998.
13. W McCune. The OTTER user's guide. Technical Report ANL/90/9, Argonne National Laboratories, 1990.
14. W McCune. Single axioms for groups and Abelian groups with various operations. *Journal of Automated Reasoning*, 10(1):1–13, 1993.
15. 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.
16. D. Redfern. *The Maple Handbook: Maple V Release 5.* Springer Verlag, 1999.
17. 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))* → *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 thm. 274 of [10]. We also need the following well known identity:

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

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 $p_1^{k_1} \dots p_l^{k_l}$, 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:

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

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.