Managing Automatically Formed Mathematical Theories

Simon Colton*, Pedro Torres*†, Paul Cairns+, and Volker Sorge×

*Department of Computing, Imperial College, London, UK. sgc,ptorres@doc.ic.ac.uk
+UCL Interaction Centre, University College, London, UK. p.cairns@ucl.ac.uk
×School of Computing, University of Birmingham, UK. V.Sorge@cs.bham.ac.uk
† Supported by Fundação para a Ciência e a Tecnologia, (SFRH/BD/12437/2003).

Abstract. The HR system forms scientific theories, and has found particularly successful application in domains of pure mathematics. Starting with only the axioms of an algebraic system, HR can generate dozens of example algebras, hundreds of concepts and thousands of conjectures, many of which have first order proofs. Given the overwhelming amount of knowledge produced, we have provided HR with sophisticated tools for handling this data. We present here the first full description of these management tools. Moreover, we describe how careful analysis of the theories produced by HR – which is enabled by the management tools – has led us to make interesting discoveries in algebraic domains. We demonstrate this with some illustrative results from HR's theories about an algebra of one axiom. The results fueled further developments, and led us to discover and prove a fundamental theorem about this domain.

1 Introduction

Automating the formation of mathematical theories has occupied Artificial Intelligence researchers for nearly 40 years. This fascination began with Lenat's inspirational – but ultimately flawed[1] – approach to mathematical concept formation via the AM and Eurisko programs [15], which formed concepts in set and number theory. Following these early attempts, methods for theory formation in particular domains were implemented, e.g., plane geometry [2], number systems (such as Conway numbers) [21] and non-associative algebras [12]. Particular attention has been paid to graph theory, with Epstein's GT program [8] providing a generic model for theory formation, and Fajtlowicz's Graffiti program [9] producing many conjectures, the proofs/disproofs of which have led to publication in the mathematical literature. More recent approaches to theory formation include further application to graph theory [19] and the approach by Pease et. al [18] to modeling the philosophy of mathematics championed by Lakatos [14].

Our contribution to mathematical theory formation has been to develop a novel descriptive machine learning algorithm, known as *automated theory formation* [3], and to apply this to discovery tasks in domains of pure mathematics, such as group theory, graph theory and number theory. Our first implementation of this technique (in the HR1 program) was written in Prolog and allowed

[1] See [1], [20] and chapter 13 of [3] for criticisms of this work.

us to investigate various concept formation and conjecture making mechanisms at a fundamental level. Our second implementation (in the HR2 program) was written in Java and addressed various disadvantages of the Prolog model. One drawback to the Prolog implementation was the lack of functionality to properly manage and present the large amounts of mathematical knowledge that was produced. In particular, we found it very difficult to extract the most interesting and important aspects of the theories produced. Hence, we have paid particular attention to implementing knowledge management tools, and HR2 boasts some fairly sophisticated ways of extracting and presenting information requested by the user. While these methods have been developed organically to meet demands from new applications, we have followed four basic principles:

- The tools implemented should be modular, hence developers should be able to augment them easily.
- The tools should enable identification of the most interesting material, even if the user's interests only come to light during or after a theory has been formed.
- Extraction of information should enable us to produce results in markup languages such as HTML, XML and LaTeX, in order to use appropriate viewers.
- The theory produced should be available for query while it is being formed.

We present here the first full description of the knowledge management techniques implemented in HR2 (hereafter referred to as simply HR). In §2, we describe the nature of the material produced by HR. We then describe the tools in HR's user interface which enable quick access of information (§3), and the more sophisticated report generation methods HR possesses (§4). We propose the hypothesis that careful use of HR's knowledge management tools enables us to extract relevant material from the theory which can lead to mathematical discovery. To add support to this hypothesis, in §5 we present the results of a series of recent sessions with HR which have led us to make some discoveries about a (relatively) little-studied algebraic domain. In §6, we summarise the management tools available in HR, and we propose improvements for future implementations of automated theory formation tools.

We will use group theory – a well known algebraic system with one operator [11] – as a running example. The operator in groups is usually denoted $*$, and this multiplies pairs of elements, x and y of a set, G, to produce a third element, $x*y$ in such a way that: (i) $*$ is associative, i.e., $\forall\, x, y \in G\, (x*(y*z) = (x*y)*z)$, (ii) there is an identity element id, such that $\forall\, x \in G\, (x*id = id*x = x)$, and (iii) each element has an inverse with respect to the identity, i.e., $\forall\ x \in G, \exists\, x^{-1} \in G$ s.t. $x * x^{-1} = x^{-1} * x = id$.

2 Theory Constituents

The HR system has many modes of operation, depending on the requirements of the user and the background information available. We focus here on two modes commonly used for forming theories about algebraic domains: (m1) the user provides only the axioms of an algebra, and (m2) the user extracts certain

results from previous sessions formed using mode (m1), most importantly the example algebras produced (which we call the *objects of interest*), and provides these as background information in new sessions. HR has two main activities which add material to the theory being produced. Firstly, HR introduces new concepts by using one of 17 production rules to take a single old concept – or two old concepts – and produce a new one. In mode m2, the initial set of concepts is supplied by the user, whereas in mode m1, the concepts are extracted from the axioms supplied. The production rules are described in detail in [5] and chapter 6 of [3]. To give a flavour of how they operate and the concepts they produce, we describe how HR can construct the concept of commutative groups. It begins by constructing the concept of commutative pairs of elements by composing the background concept of group multiplication with itself. That is, given this background concept:
1.$[a, b, c, d] : b, c, d \in a \wedge b * c = d$
HR produces the following new concept using the *compose* production rule:
2.$[a, b, c, d] : b, c, d \in a \wedge b * c = d \wedge c * b = d$
HR then uses the *exists* production rule to generalise concept 2 further:
3.$[a, b, c] : b, c \in a \wedge \exists\, d \text{ s.t. } (b * c = d \wedge c * b = d)$
Finally, HR uses its *forall* rule to produce the concept of commutative groups:
4.$[a] : \forall\, b, c\, (b, c \in a \rightarrow \exists\, d \text{ s.t. } (b * c = d \wedge c * b = d))$

This construction highlights the fundamental information that HR records about concepts, namely the concept identification number, the scope of the concept, i.e., the tuple in square brackets which represents objects to which the concept applies, and the definition of the concept which tuples must satisfy. HR also records a plethora of other information about each concept produced. This includes (a) the ground instances of tuples which satisfy the concept definition, i.e., the success set of the definition (b) the construction history of the concept (c) the way in which the concept categorises the examples, and (d) various numerical values which measure how interesting the concept is, e.g., the *applicability* of a concept calculates the proportion of objects of interest which appear in the concept's success set, and the *novelty* of a concept calculates the reciprocal of the number of times the categorisation of examples afforded by that concept is also the categorisation for another concept. As discussed in [6], while the measures of interestingness drive HR's heuristic search, they are also useful for sorting and pruning the concepts it produces. In particular, we often find that concepts which score highly for novelty are the most interesting in a theory.

HR's second main activity is to make conjectures about the concepts. It does this empirically, by noticing patterns in the success sets of the concepts. For instance, if HR invents a new concept with exactly the same examples as a previous one, it makes the conjecture that the definitions of the two concepts are equivalent. Moreover, whenever a new concept is added to the theory, HR checks to see whether the new concept's examples are a subset or superset of the examples of a previous concept, and makes implication conjectures accordingly. Given an empirically formed conjecture, HR extracts simpler conjectures from it, i.e., from an equivalence conjecture, it extracts two implication conjectures, from an implication conjecture, it extracts Horn clause implicates (where a conjunction of premises implies a single goal), and from Horn clause implicates, it extracts prime implicates, where no proper subset of the premises implies the goal. In addition, HR uses the Otter theorem prover [16] and the Mace model generator [17] to prove/disprove conjectures. Otter therefore introduces a third type of theory constituent, namely proofs, and Mace introduces a fourth, namely new objects of interest (e.g., groups) which act as a counterexample to a non-theorem.

HR's conjecture making functionality is described in detail in [4]. As an illustration, in mode m1, HR is given no example groups, so it first forms the conjecture that there are actually no groups. Mace disproves this by supplying the trivial group as a counterexample. HR then conjectures that the only element in a group is the identity element and Mace supplies the cyclic group of order two as a counterexample. Later on, HR invents the concept of idempotent elements, i.e., elements, x, for which $x * x = x$. Given that the success set of this concept is exactly the same as that for the background concept of identity elements, HR makes this conjecture: $\forall\, G, \forall\, b \in G\, (b * b = b \leftrightarrow b = id)$. Otter proves this easily, and HR extracts the two obvious implication conjectures, which are already prime implicates, so no further processing is performed.

In summary, theories produced by HR include (a) objects of interest, which are introduced by the user or as counterexamples produced by a model generator (b) concepts which categorise the objects of interest (c) conjectures which relate the concepts, and (d) proofs of the conjectures. We see that there is a high degree of cross referencing between the different types of theory constituent. In addition, the volume of this material is quite large. For instance, in a mode m1 session, S, of 5000 theory formation steps lasting 7163 seconds in group theory, HR produced 9 objects of interest (groups), 306 concepts and 3017 conjectures, of which 2941 were supplied with Otter proofs. In the next two sections, we describe how HR manages this information.

3 Front-end Management Utilities

The graphical user interface to HR is extensive, consisting of more than 300 widgets (check boxes, buttons, lists, text fields, etc.,) spread over 20 screens. Nine of these screens enable the user to set up HR at the start of a theory formation session, and ten screens enable the user to probe the theory during (as HR is multi-threaded) and after theory formation. The final screen enables the user to record changes to any of the other screens into text (macro) files. Running one of these macro files enables the widget changes and button clicks to be repeated at the start of a new session, which is a useful tool. Of the ten theory-probing screens, three are devoted to directly presenting information about the theory constituents, i.e., there is a separate screen for the objects of interest, the concepts and the conjectures in the theory. There is no separate screen for the proofs, as these are attached to the conjectures which they prove.

In each of the three theory constituents screens, there is a main text area where information about a chosen theory constituent is displayed. There are also four lists which enable the user to be specific about the information presented. These lists are: (a) the constituent list, which enables the user to choose which constituent to look at (b) the details list, which enables the user to turn on and off various pieces of information about the chosen constituent (c) the pruning list, which enables the user to temporarily hide constituents which do not satisfy certain criteria, and (d) the sorting list, which enables the user to order the constituents according to various measures of interestingness. In combination, these lists enable quite specific pinpointing of aspects of the knowledge HR generates. For instance, in the concept-probing screen, there are 59 details to choose from, 23 ways to prune the concepts and 19 ways to sort the concepts. HR also provides many ways to cross reference the theory constituents, for instance by providing the list of conjectures which involve a concept in the details for that concept. HR also provides a number of other tools for probing the theory constituents directly, e.g., a text search facility in the concepts screen. HR can present concept definitions and conjecture statements in plain text, in Otter format, in Prolog format, and in TPTP format [22]. It also has an ability to simplify concept definitions, e.g., it would simplify the Otter-style definition of commutative groups from §2 above to: $4.[a] : \forall\, b, c \in G\, (b{*}c = c{*}d)$. In addition, HR uses Dot [13] to present graphical construction histories of concepts and conjectures.

To illustrate usage of these screens, after the theory formation in session S described in §2 above, we looked at the conjectures, narrowed down our view to just proved equivalences, and sorted them by the length of the proof produced by Otter. We identified this conjecture as having the longest proof (68 steps):

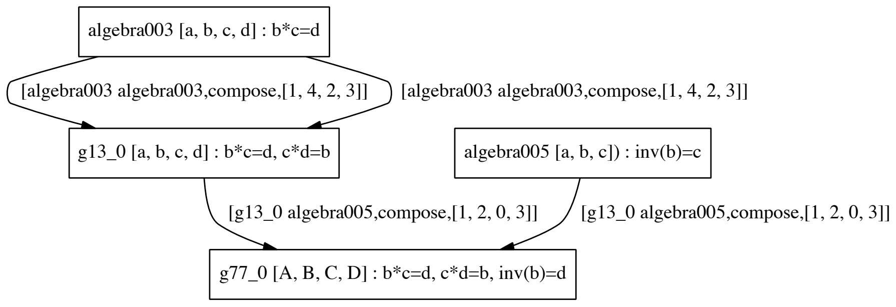
$$\forall bcd\,((b{*}c = d \wedge c{*}d = b \wedge b^{-1} = d) \leftrightarrow (b{*}b = c \wedge c{*}b = d \wedge c{*}d = b \wedge (\exists\, e\,(e{*}c = d))))$$

We cross referenced the left hand concept of this equivalence conjecture and found that it had a normalised novelty value of 0.83 (hence it was quite novel). We also generated the construction history graph for the concept, as shown in figure 1. We noted that it was constructed using only the compose production rule.

4 Report Generation Techniques

An advantage of the HR1 Prolog implementation was being in an interpreted environment where bespoke queries could be constructed. This functionality is crucial, because theory formation can take many hours, and often the exact criteria which we need to pinpoint the theory constituents of interest only become clear on consideration of the formed theory. One possibility to enable bespoke queries at run-time would be to output the theory to a database, and use SQL to query it. Another possibility would be to output a Prolog representation of the theory (which HR can do) and query it using Prolog. Another possibility would be to write and compile bespoke Java classes and use class-loading at run-time.

Our solution offers, we believe, more flexibility than any of these alternatives. We have implemented a Java interpreter which uses the Java reflection



algebra003 [a, b, c, d] : b*c=d
[algebra003 algebra003,compose,[1, 4, 2, 3]]
[algebra003 algebra003,compose,[1, 4, 2, 3]]
g13_0 [a, b, c, d] : b*c=d, c*d=b
algebra005 [a, b, c]) : inv(b)=c
[g13_0 algebra005,compose,[1, 2, 0, 3]]
[g13_0 algebra005,compose,[1, 2, 0, 3]]
g77_0 [A, B, C, D] : b*c=d, c*d=b, inv(b)=d

Fig. 1. Construction history for the group theory concept appearing in the theorem with the longest Otter proof.

mechanism to execute scripts supplied by the user at run-time. Each script is supplied with pointers to certain objects, in particular to the theory object that HR has produced. In effect, this enables the user to write a script to query in any way, any aspect of the theory. At present, the interpreter is able to handle simple arithmetic and string manipulation, for-loops, if-statements, and to call any public method or access any public field of any class, including those imported from the core Java API (which enables, amongst many other things, file input/output). As an example, the script in figure 2 prints to the standard output an XML file with the definition of each concept along with the sum of the concept's applicability and novelty values.

```
Vector concepts = theory.concepts;
for (int c_num = 0; c_num < concepts.size(); c_num++){
  Concept concept = concepts.elementAt(c_num);
  System.out.println("<concept>");
  System.out.println("   <definition>");
  System.out.println(concept.writeDefinition("otter");
  System.out.println("   </definition>");
  System.out.println(concept.write("   <sum>"));
  System.out.println(concept.applicability + concept.novelty);
  System.out.println(concept.write("   </sum>"));
  System.out.println("</concept>")
};
```

Fig. 2. Example script for producing XML output

We take full advantage of the ability to execute scripts at run-time. In addition to a screen where the user can write scripts with the output directed to a text area, there are also text boxes on the concepts, conjectures and objects of interest screens described above, which enable the user to type in Java code to act on the theory constituent being examined. Also, the macros described in section 3 above can be embellished with Java code. Moreover, we use the interpreter inside the reports, statistics and react screens, as described below.

The report screen provides more support to the user in writing report generating scripts such as those in figure 2 above. In particular, the screen allows the user to write and save report scripts, to choose one or more scripts to run simultaneously, to run reports during/after theory formation, and to dictate after how many new theory formation steps the reports should be run. Each script is passed the theory object, and the user must specify which theory constituents (concepts, equivalences, implications, implicates, objects of interest, etc.) the script will report on. Looping through the constituents is organised internally, so the user only needs to supply Java code for outputting information either to the screen or to a file. The user can also dictate how many times the loop is performed, and can specify code to be run at the start/middle/end of each loop. This allows cumulative values to be calculated – e.g., the average applicability of concepts – and reported. We have written fairly sophisticated scripts which produce hyper-linked web pages. For example, one script lists the proved prime implicates present in the theory, with each one hyper-linked to further information, including the proof of the prime implicate.

In the statistics screen, the user is able to tabulate numerical and textual information about the theory constituents. In particular, there are three lists, from which the user can choose: the set of theory constituents to tabulate information for; the public fields for those constituents; and the public methods to be run for the constituents. There is also a text box into which the user can type Java code to apply to each of the constituents. Moreover, HR can interface with the GnuPlot program to plot information on a graph. For instance, in figure 3, we plot the average novelty of concepts as the theory progresses. Note that the graph exhibits a saw-tooth behaviour, with peaks whenever a highly novel concept is formed which decline as the concept loses its novelty. We have found such analyses to be very useful during theory formation. The user can also choose to see run-time statistics, in order to identify bottlenecks in the theory formation process.

In the react screen, the user can write scripts to be run at one of seven key points during a theory formation step (e.g., after a new concept has been added to the theory). Often, this functionality is used to alter the search according to the results of the step. However, we have often inserted report generation scripts which react to certain events, e.g., to notify us of certain highly interesting concepts as soon as they are formed.

5 Enabling Mathematical Discovery - Illustrative Results

We demonstrate here how study of the three major theory constituents (objects of interest, concepts and conjectures/theorems) that HR produces can lead to novel discoveries in pure mathematics. We look at a (relatively) little-studied algebraic structure which we refer to as star algebras.[2] These algebras have a single axiom, which resembles associativity: $\forall\, x, y, z\, ((x * y) * z = y * (z * x))$.

[2] Initial conversations with J.D. Phillips inspired our study of this domain. Unfortunately, we have not been able to ascertain why they are called star algebras.

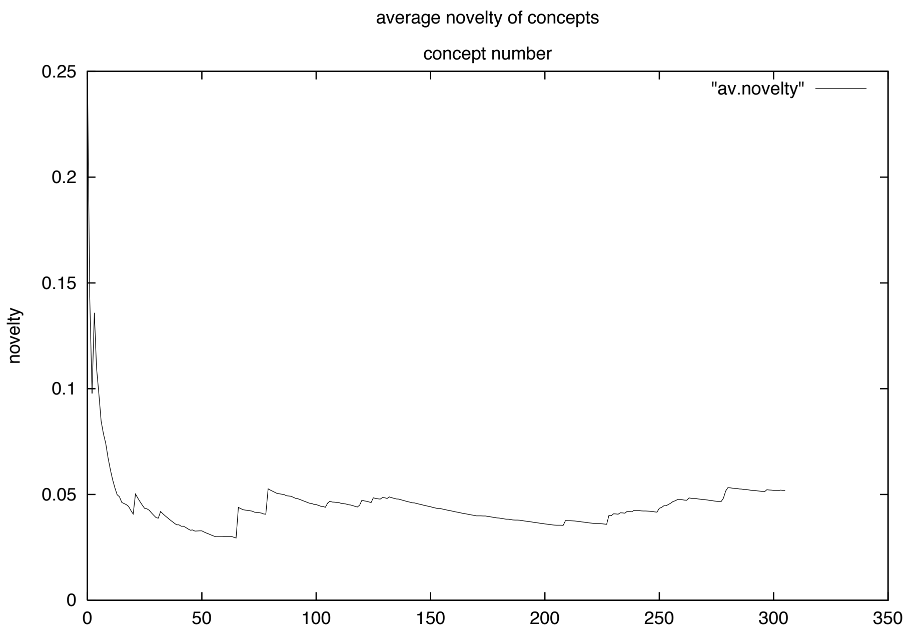


Fig. 3. Average Novelty of Concepts during Theory Formation

5.1 Identity Results

We first formed a theory in mode m1 (from the axioms alone) which enabled HR to generate 39 star algebras. These were then extracted and used in later theory formation sessions (mode m2). We then looked at the proved prime implicates generated by HR and we discovered some results about left and right identities. In particular, HR highlighted the fact that: $\forall a, b\,(b * a = a \rightarrow a * b = a)$. Paraphrased, this states that, in star algebras, any element which is a left identity for another element is also a right identity for that element. We also used the first order generic production rule [23] to specify closure under multiplication as interesting to study. Using this, HR conjectured – and Otter proved – that the elements which have a left identity are closed under multiplication, i.e., the product of two elements which both have a left identity is also an element which has a left identity.

This result also holds for elements with a right identity. However, on looking at the sub-star algebras produced by taking all elements with a right identity, we noticed that they were always the trivial algebra. Note that HR's embed-algebra production rule enabled us to look at the sub-algebras. This led us to form the incorrect hypothesis that right-identity sub-algebras are always trivial. Given that we had evidence of non-trivial left-identity sub-algebras, our hypothesis is false, as, if we take any star-algebra S and produce S' by writing $b * a$ in place of $a * b$ in the multiplication table, then S' is still a star algebra (proof omitted).

Clearly, left identities in S are right identities in S', hence there are non-trivial right-identity sub-algebras.

5.2 Idempotency Results

HR also used the first order generic production rule to discover and prove that idempotent elements (such that $x * x = x$) are closed under multiplication. We cross-referenced concepts from this conjecture, and narrowed our attention to star algebras which specialise the domain. In particular, we examined the concept of idempotent star algebras (i.e., star algebras for which $\forall x\,(x * x = x)$). We cross-referenced this further by looking at some of the examples which satisfied the specialisation. Their multiplication tables were as follows:

	0	1	2
0	0	0	0
1	0	1	0
2	0	0	2

	0	1	2	3
0	0	0	0	0
1	0	1	0	0
2	0	0	2	0
3	0	0	0	3

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	0	0	0
2	0	0	2	0	0
3	0	0	0	3	0
4	0	0	0	0	4

We found that, in all the examples of the idempotent concept, all the non-diagonal entries on the multiplication table were the same element (zero). Given this, we hypothesised and proved that all similar constructions (of any size) produce star algebras (proof omitted). Note that we used Mace to disprove the hypothesis that such star algebras are the only idempotent ones. We used the classification system described in [7] to prove that idempotent star algebras for which all the non-diagonal entries are the same element (not necessarily zero) form an isomorphism class for sizes 6, 7 and 8. This gave us good empirical evidence and the confidence to prove that these specialisations characterise a *family* of star algebras, i.e., for every order n, these star algebras form an isomorphism class (c.f., cyclic groups in group theory, etc.).

5.3 Canonical Examples

In a separate set of experiments, we concerned ourselves with finding canonical examples of star algebras. Broadly speaking, we were looking for star algebras with certain properties which were not bestowed upon them by the fact that they satisfied another axiom set. For instance, many properties of certain star algebras are true purely because they are commutative and/or associative. Hence we were more interested in looking at non-associative, non-commutative star-algebras. We generalised this notion of canonical examples, and wrote a new measure of interestingness with respect to the objects of interest, called ImpOut (shorthand for implication outlier), as described below.

Given concepts $p(a_1, \ldots, a_n)$ and $q(a_1, \ldots, a_n)$, HR makes an implication conjecture $p \rightarrow q$ whenever the set of examples for concept p is a subset of the set of examples for q. Hence, any example satisfying the definition of p will also satisfy the definition of q. The ImpOut measure was designed to capture the idea of objects of interest which which are examples of concept q but not by virtue of being examples of concept p. More rigorously, ImpOut is a function associating a real number in the interval $[0, 1]$ to each object of interest in the theory. The corresponding ImpOut value for object of interest E is computed by considering an enumeration of all HR implication conjectures $\{p_i \rightarrow q_i\}_i$ and dividing the number of those implications for which E satisfies q_i but not p_i by the number of implications for which E satisfies q_i.

When using the ImpOut measure to sort the 39 star algebras in the theory, we noticed that the least interesting with respect to this measure were those which had a repeated row or column, such as examples A and B below, and the most interesting had no repetition, such as examples C and D below:

A	0	1
0	0	0
1	0	0

B	0	1	2	3
0	3	3	1	3
1	3	3	3	3
2	3	3	1	3
3	3	3	3	3

C	0	1	2
0	1	2	0
1	2	0	1
2	0	1	2

D	0	1	2	3
0	3	2	1	0
1	2	0	3	1
2	1	3	0	2
3	0	1	2	3

In addition, we ran another theory formation session in mode m2, where we specialised the axioms to non-commutative, non-associative star algebras. HR generated 5 examples, and in each one, we noticed a repeated row. Analysis of these and other results led us to the following hypothesis: the only non-associative, non-commutative star algebras are those with a repeated row or column in their multiplication table. We tried to prove this using the Vampire, Otter and E provers, but each failed to prove it within 72 hours of processing. We also used the Mace and Finder model generators to exhaust the search for counterexamples up to size 10, but we found none. This fueled our interest, and we eventually proved a more general result by hand, as shown below.

Definition.
An algebra is said to be *k-nice* if the product of any k elements is the same regardless of bracketing or the order of the elements in the product. (This definition is taken from [10]). For instance, commutative, associative algebras are 3-nice, because multiplying a triple of elements in any way always produces the same product.

Definition.
An algebra is said to be *redundant* if there exist two distinct elements x and y in S, such that, for every e in S, we have: $x * e = y * e$ and $e * x = e * y$.

Theorem.
For $k > 3$, every k-nice non-commutative algebra S is redundant.

Proof.
Given a pair of elements (α, β) from S, we define the following recursive algorithm: if it is possible to find an element x of S such that $x * \alpha \neq x * \beta$, then recurse with the pair $(x * \alpha, x * \beta)$. Otherwise, if it is possible to find an element y of S such that $\alpha * y \neq \beta * y$, then recurse with the pair $(\alpha * y, \beta * y)$. Otherwise, output the pair (α, β) and stop.

The first point to notice is that, if we let $\alpha = a * b$ and $\beta = b * a$ for some $a, b \in S$, then the algorithm will terminate. To see this, we note that the algorithm simply multiplies both elements of the pair by some other element at each recursion. Therefore, if the algorithm hasn't already terminated by recursion number $k-1$, then both elements in the pair will be representable as the same set of $k-1$ elements multiplied together, albeit in a different order (because of the reversal of the a and b in α and β). By definition of S being k-nice, if we multiply both elements in the pair by the same element (either on the right or left), then the product – of k elements – will be the same. Hence, we will not be able to find an element which multiplies on the left or right of the pair to give distinct elements, and the algorithm will terminate.

The second point to notice is that, if $\alpha \neq \beta$, then the output of the algorithm will likewise be a pair of distinct elements. This is by definition of the algorithm: at each recursion, the next pair is constructed specifically to be distinct, and the output of the algorithm is just the input to the final recursion.

Now, given that S is non-commutative, we can find $x, y \in S$ such that $x * y \neq y * x$. As shown above, if we input $(x * y, y * x)$ to the algorithm, it will terminate and output a distinct pair of elements (x', y'). These will be such that $\forall e \in S, (e * x' = e * y' \wedge x' * e = y' * e)$. Hence, S is redundant. □

Corollary.
The set of non-redundant star algebras is exactly the set of non-redundant commutative and associative algebras.

Proof.
In [10], Hentzel et. al. prove that star algebras are 5-nice. Therefore, the theorem above implies that non-commutative star algebras are redundant. Taking the converse, we conclude that any non-redundant star algebra, S, is commutative. S will also be associative because $\forall\, a, b \in S((a*b)*c \overset{\text{com}}{=} c*(a*b) \overset{\text{star}}{=} (b*c)*a \overset{\text{com}}{=} a*(b*c))$. In addition, as commutative, associative algebras are 3-nice, they clearly satisfy the star-algebra axiom. □

Returning to the question of canonical examples of star algebras, we can now conclude that the only non-associative, non-commutative star-algebras are those with repeated rows and columns, as the data from HR suggested. Indeed, the set of non-redundant star-algebras is the set of associative, commutative algebras, hence non-redundant star-algebras are not worthy of further study. Moreover, given that the theorem above states that non-commutative star algebras have the same pair of rows and columns repeated, we can define the following reduction algorithm which will transform a redundant star-algebra into a non-redundant one.

Reduction Algorithm.
Suppose $(S, *)$ is a redundant algebra with distinct α and β satisfying the redundancy condition $\forall e\, (\alpha e = \beta e \wedge e\alpha = e\beta)$. We define the *reduction of* $(S, *)$ *with respect to* (α, β), denoted $\rho_{(\alpha,\beta)}(S, *)$, as the algebra $(S \backslash \{\beta\}, *')$, where $*'$

is defined for all $a, b \in S\backslash\{\beta\}$ as

$$a *' b = \begin{cases} \alpha & \text{if } a * b = \beta \\ a * b & \text{otherwise} \end{cases} .$$

```
**while**( $(S, *)$ is redundant )
    find $(\alpha, \beta)$ such that $\alpha \neq \beta \wedge \forall e\,(\alpha e = \beta e \wedge e\alpha = e\beta)$
    $(S, *) \leftarrow \rho_{(\alpha,\beta)}(S, *)$
**end**
```

Hence, we see that the only canonical examples worth studying are trivially reducible to commutative, associative algebras, which largely draws a line under this avenue of investigation.

6 Conclusions and Future Work

We have presented the knowledge management tools implemented in the HR system which enable us to cherry pick and cross-reference the most interesting information from the theories it forms. These tools have been developed according to four principles. Firstly, as HR is multi-threaded, the user is able to query the theory as it is being produced, and can stop or alter the search accordingly. Secondly, due to the Java interpreter in HR, the user can construct bespoke queries and generate novel types of report during and after a theory has been formed. Thirdly, the tools are modular, as demonstrated by our addition of the ImpOut measure of interestingness. Fourthly, thanks to the embedding of the interpreter in various screens, and HR's ability to write concept definitions and conjecture statements in a variety of syntaxes, the tools are able to output information in LaTeX, XML, HTML, Prolog, TPTP etc., format, enabling the usage of appropriate viewers, in addition to the output of graphical summaries of the theory and of individual theory constituents. In future implementations, we intend to build on the Java interpreter, report generator and cross-referencing of material between screens, as we have found these the most useful of HR's knowledge management tools.

We have demonstrated the flexibility and utility of the knowledge management tools during an investigation which led to some interesting mathematical results. In one sense, the investigation of star algebras from the perspective of canonical examples led to a negative result. That is, we have proved that any star algebra which is non-associative and non-commutative (properties which a canonical example should have), has these properties purely by virtue of having a redundant row and column. However, the investigation led to a more general – and surprising – result: that any k-nice non-commutative algebraic structure has redundancy. While HR and it's knowledge management tools cannot take credit for the discovery and proof of this result, it is clear that they played their part in this investigation. We believe that such interactive use of mathematical theory formation systems can enhance mathematical research, and we plan to make our techniques more powerful, and our interfaces and knowledge management tools more flexible to appeal to research mathematicians.

Acknowledgments

We would like to thank the anonymous reviewers for their interesting and instructive comments.

References

1. M Anderson. A critical evaluation of Lenat's AM program. Technical Report TR 89-09-19, Department of Computer Science, University of Washington, 1989.
2. R Bagai, V Shanbhogue, J Żytkow, and S Chou. Automatic theorem generation in plane geometry. In *LNAI 689*. Springer, 1993.
3. S Colton. *Automated Theory Formation in Pure Mathematics*. Springer, 2002.
4. S Colton. The HR program for theorem generation. In *Proceedings of CADE*, 2002.
5. S Colton, A Bundy, and T Walsh. Automatic identification of mathematical concepts. In *Proceedings of ICML*, 2000.
6. 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.
7. S Colton, A Meier, V Sorge, and R McCasland. Automatic generation of classification theorems for finite algebras. In *Proceedings of the IJCAR*, 2004.
8. S Epstein. On the discovery of mathematical theorems. In *Proceedings of the International Joint Conference on Artificial Intellignce*, 1987.
9. S Fajtlowicz. On conjectures of Graffiti. *Discrete Mathematics 72*, 23, 1988.
10. I Hentzel, D Jacobs, and S Muddana. Experimenting with the identity (xy)z = y(zx). Technical report, Clemson University, 1993.
11. J Humphreys. *A Course in Group Theory*. OUP, 1996.
12. D Jacobs. The Albert non-associative algebra system: a progress report. Technical report, Clemson University, 1994.
13. E Koutsofios and C North. Dot user's guide. Technical report, AT+T Bell Labs, Murray Hill, NJ, 1998.
14. I Lakatos. *Proofs and Refutations: The logic of mathematical discovery*. Cambridge University Press, 1976.
15. D Lenat. AM: Discovery in mathematics as heuristic search. In *Knowledge-Based Systems in Artificial Intelligence*. McGraw-Hill, 1982.
16. W McCune. The OTTER user's guide. Technical Report ANL/90/9, Argonne National Laboratories, 1990.
17. W McCune. A Davis-Putnam program and its application to finite first-order model search. Technical Report MCS-TM-194, Argonne National Labs, 1994.
18. A Pease and S Colton. Modelling Lakatos's philosophy of mathematics. In *Proceedings of the Second European Conference on Computing and Philosophy*, 2004.
19. H Pistori and J Wainer. Automatic theory formation in graph theory. In *Argentine Symposium on Artificial Intelligence*, pages 131 – 140, 1999.
20. G Ritchie and F Hanna. AM: A case study in methodology. *Artificial Intelligence*, 23, 1984.
21. M Sims. *IL: An Artificial Intelligence approach to theory formation in mathematics*. PhD thesis, Rutgers University, 1990.
22. G Sutcliffe and C Suttner. The TPTP problem library: CNF release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.
23. P Torres and S Colton. Applying model generation to concept formation. In *Proceedings of the Automated Reasoning Workshop*, 2006.