Automatic identification of mathematical concepts

Automatic Identification of Mathematical Concepts

Simon Colton SIMONCO@DAI.ED.AC.UK
Alan Bundy BUNDY@DAI.ED.AC.UK
Division of Informatics, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, United Kingdom

Toby Walsh TW@CS.YORK.AC.UK
Department of Computer Science, University of York, Heslington, York Y010 5DD, United Kingdom

Abstract

The HR program by Colton et al. (1999) performs theory formation in mathematics by exploring a space of mathematical concepts. By enabling HR to determine when it has found a particular concept, and by adding a forward looking mechanism, we have applied HR to the problem of identifying mathematical concepts. We illustrate this by using HR to identify and extrapolate integer sequences and by performing a qualitative comparison with the machine learning program Progol.

1. Introduction

Extrapolating integer sequences such as $1, 4, 9, 16 \ldots$ is an intelligent activity requiring both understanding and creativity. While there have been attempts in Artificial Intelligence to automatically extrapolate sequences, presently the state of the art is to use a large online[1] database, the Encyclopedia of Integer Sequences. Extrapolating integer sequences generalises to the problem of automatically identifying a property of a set of mathematical objects (the example set) which distinguishes the objects from a larger class.

The HR program (Colton et al., 1999) performs theory formation by exploring a space of mathematical concepts in domains such as finite group theory, graph theory and number theory. The space is characterised by the initial set of concepts supplied by the user, from which all subsequent concepts are built, and by the seven production rules which turn old concepts into new ones. To derive the production rules, we determined some general properties common to many concepts across domains and implemented production rules which can turn old concepts into new ones which have these properties. Given some initial concepts as background information, HR uses a heuristic search to choose which old concepts to apply which production rules to and explores the space in this manner.

If the user supplies positive and negative examples for a concept, and HR invents a definition which all the positive examples satisfy, but the negative examples do not, we say HR has identified the concept. In number theory, we have restricted HR's abilities to only identifying sequences which are types of number, such as prime numbers. We hope to cover other sequence types such as functions in the near future. If the user supplies an increasing sequence: $a_1, a_2, \ldots, a_n$ then HR assumes that the negative examples are those integers between 1 and a_n which do not appear in the sequence. For example, when HR is given this sequence extract:

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31$$

it re-invents the concept of integers with exactly two divisors (prime numbers). All the integers in the sequence have this property but the integers $1, 4, 6, \ldots, 30$ do not, so HR has learned the concept. To supply the next number in the sequence HR simply generates integers larger than a_n until the next one which satisfies the definition is found, in this case 37.

HR is designed to perform machine discovery and has successfully invented new integer sequences (Colton et al., in pressa). The purpose of this paper is to detail the application of HR to a different problem: the identification of user supplied concepts. Exploratory search is not well suited to such learning problems, so we have implemented a forward looking mechanism to improve efficiency. This works by looking up to three steps ahead in the space to decide whether a path will lead to the required concept. We discuss how concepts are represented, the production rules HR employs and the forward looking mechanism. To assess the system, we present some initial results from integer sequence extrapolation and compare HR to other systems, including the machine learning program Progol.

[1] `http://www.research.att.com/~njas/sequences`

2. Representation of Concepts

HR is supplied with background knowledge in terms of some initial concepts upon which all new concepts are based. These concepts are either objects of interest, such as graphs, groups or integers, ways to finitely decompose the objects into subobjects, or relations between the subobjects. Table 1 details the subobjects and relations usually provided for each domain.

Table 1. Initial concepts supplied by the user

Domain	Subobject	Relation
graph theory	nodes, edges	adjacency of nodes, node on edge
group theory	elements	identity, inverse, group operation
number theory	divisors, digits, binary digits	addition, $\leq$, multiplication

Each concept describes a property of tuples of objects. The tuple is always an object of interest followed by subobjects. An example is prime divisors of an integer: a property of pairs of integers, (n, p) stating that p is a prime divisor of n. Additionally, there may be numerical values calculated, for example the concept of tuples (G, x) where G is a graph and x is the number of nodes. Every concept is represented with:

[1] A **data table** of tuples of objects which satisfy the definition of the concept. For example, for the integers 1 to 5, the concept of multiplication has these tuples:

$[1,1,1]$ $[2,1,2]$ $[2,2,1]$ $[3,1,3]$ $[3,3,1]$
$[4,1,4]$ $[4,2,2]$ $[4,4,1]$ $[5,1,5]$ $[5,5,1]$

where the integers in the final two columns multiply to give the integer in the first column.

[2] A set of **predicates** which are true of the objects in each row of the data table. For example, the concept of multiplication discusses triples of integers, $[n, a, b]$ and the user supplies 6 predicates:

- $integer(n), integer(a), integer(b)$
  $divisor(n, a), divisor(n, b), multiplication(n, a, b)$

These predicates specify the multiplication property of this concept and also highlight that the integers being multiplied together are divisors of their product.

Given a new concept, C, HR identifies its **sub-concepts**, which are those other concepts with a subset of the predicates of C. For instance, the concept of divisors in number theory is a sub-concept of multiplication, as the integers being multiplied are divisors of their product. We call sub-concepts where each variable of C appears in a predicate of the sub-concept **generalisations**. For example, divisors are a generalisation of prime divisors.

3. Production Rules

Instead of studying how mathematicians invent new concepts, we have looked at the concepts themselves to extract shared general properties. We have turned each observed property into a production rule which takes a concept without the property and produces a concept with it. We have implemented seven rules, each of which generates a data table and predicates for the new concept. Each rule also generates a set of concept-dependent parameterisations, with each parameterisation specifying a different construction using the rule. We motivate each rule below with examples from mathematics, and detail how it is parameterised and how it produces new data tables and predicates. The seven rules are not meant to be exhaustive, and we suggest two more in Section 7.

3.1 The Negate Production Rule

The negation of one concept is often incorporated into the definition of another, e.g., odd numbers are those not divisible by 2. Similarly, groups which are not soluble are very important in group theory. Given a concept, C, there is one way to negate it for each of its generalisations, G, and the set of generalisations comprises the parameterisations for this rule. To construct the predicates of the new concept, the negate rule finds any predicates of C which aren't predicates of G, and negates them. It then adds the negated predicates to those of G to produce the new set. For example, the concept of pairs of divisors of an integer[2]:

$$[n, a, b] : a|n \wedge b|n$$

is a generalisation of the multiplication concept:

$$[n, a, b] : a|n \wedge b|n \wedge a \times b = n$$

Hence to negate multiplication, HR negates just the multiplication predicate and adds it to the predicates from the generalisation, giving:

$$[n, a, b] : a|n \wedge b|n \wedge a \times b \neq n$$

The data table for the new concept is constructed by taking those tuples in the data table for G that are not present in the data table for C. In our example, the new data table will contain tuples such as $[6, 2, 2]$ and $[10, 1, 5]$ where the last two numbers are divisors of the first integer, but their product is not the integer.

3.2 The Forall and Exists Production Rules

Looking at Abelian groups, where all elements commute, complete graphs, where all nodes are adjacent and repdigit integers, where all digits are equal, we see that objects where every pair of subobjects satisfies a

[2] The definition for this concept is to be read: "triples $[n, a, b]$ such that a divides n and b divides n."

relation are often extracted into a new concept. In general, objects where every tuple of subobjects have a particular property are of interest.

To turn this observation into the forall production rule, we noted that information about which specific subobjects have the property is replaced by a $\forall$ quantification, ie., all objects have the property. The rule is then parameterised by the set of variables to quantify over. For instance, the parameters $[2,3]$ indicate that the second and third variables should be quantified over. Given a concept, C, the predicates for the new concept are the predicates not involving the quantified variables, and an additional quantified predicate. The quantified predicate is constructed by first finding a sub-concept, S, which involves all the variables to be quantified. It is then constructed in the form: $\forall\ a, b, c$ s.t. $X(a,b,c), (Y(a,b,c))$, where X is the conjunction of the predicates of S and Y is the conjunction of the predicates of C which are not predicates of S.

This is better explained by example: suppose we use parameters [2] and the concept of prime digits:

$$[n, a] : a \text{ is a digit of } n \wedge a \text{ is prime .}$$

The only sub-concept of this is the concept of digits. Hence the new concept will have a quantification over the digits of n, stating that they are all prime:

$$[n] : \forall a \text{ s.t. } a \text{ is a digit of } n, (a \text{ is prime}).$$

This is the concept of integers with all prime digits. To produce a data table for the new concept, the data table of the sub-concept S is found and HR looks at every tuple in it for a particular object of interest. If all the tuples are also found in the data table for C, a new tuple is constructed containing just the object and added to the data table for the new concept.

Sometimes objects with all subobjects satisfying a relation are rare and it may still be interesting to find an object which has at least one such subobject. The exists production rule generates concepts with this kind of existential quantification. The construction of new concepts by the exists production rule is similar to that performed by the forall rule.

3.3 The Match Production Rule

Square numbers, self inverse elements in groups and loops in graphs from a node to itself are all concepts where some objects or subobjects relate to themselves. The match production rule implements a method to construct concepts with this nature. Each parameterisation for this rule details how to find tuples where some of the objects are equal, with the proviso that they are not different types of objects, such as nodes and edges of graphs. For instance, the parameters $[1,2,2]$ specify that tuples where the last two subobjects are equal are to be extracted into a new concept. As the objects will be equal, the corresponding variables in the predicates must also match. Therefore predicates for the new concept are constructed by taking the predicates for the old concept and matching the variables. For example, given parameters $[1,2,2]$ and the multiplication concept:

$$[n, a, b] : a|n \wedge b|n \wedge a \times b = n$$

if we extract all those tuples for which the last 2 objects are equal, we get this concept:

$$[n, a] : a|n \wedge a \times a = n$$

which is a construction made on the way to defining square numbers. Note that the arity of the concept has been reduced, ie. the predicates are now binary, as the third variable is the same as the second. To construct a data table for the new concept, HR extracts all those tuples where the objects in the columns match as prescribed by the parameters. It then discards the repeated columns relating to the repeated variables.

3.4 The Size and Split Production Rules

The *number* of divisors of a integer is the well known τ-function and the *size* of the centre of groups is an important group theory notion. Also, groups with one central element and integers with *two* divisors are well known concepts (symmetric groups and prime numbers respectively). This indicates that taking the size of a set of subobjects, and identifying objects with a set of a particular size are two general ways to construct mathematical concepts.

Given a concept, C, the size production rule calculates set sizes. Similar to the forall rule, information about specific subobjects is replaced by a summary of how many have a particular property. So, this rule is parameterised by sets of variable numbers such as $[2,3]$ which specify that subobjects appearing in the 2nd and 3rd columns of the tuples for C are to be counted. The predicates for the new concept are produced by first finding all predicates involving no variables for the sub-objects to be counted. These are added to the new set, along with an additional predicate involving a new variable, say x. The new predicate will be of the form: $x = |\{(a,b,c) : Z(a,b,c)\}|$, where Z is the conjunction of the set of predicates which involve the variables for the sub-objects being counted.

For example, if we start with the concept of division: $[n, a] : a|n$, we can count the number of divisors of n. Removing the predicates which involve divisors leaves only the sub-concept of integers. A new letter is generated, x, and the predicate to add will be $x = |\{a : a|n\}|$. This produces the τ-function:

$$[n, x] : x = |\{a : a|n\}|$$

The new data table is constructed by finding the subconcept, S, with the predicates of C involving variables not appearing in the new set size predicate. Then, for each tuple in the data table of S, the number of times it appears in the data table for C is counted. Finally, this coefficient is added to the tuple to produce a tuple for the new data table.

The split production rule takes a concept generated by the size production rule and finds those objects where the set size is either 0, 1 or 2 (or a higher number, depending on the user's preferences). To produce new predicates, it simply replaces the variable representing the set size with the value the set size must be. For example, taking the τ function above, and replacing the x by 2, we get the concept of prime numbers:

$$[n] : 2 = |\{a : a|n\}|$$

This action reduces the arity of the concept, as one of the variables has changed to a fixed number.

This rule can also fix subobjects to be particular values. For example, if it extracts all those integers where 2 is a divisor, it produces even numbers. The parameterisations specify which columns of the tuples to look for which values in. For example, to get prime numbers, we need to look for the number 2 in the second column of the τ-function tuples. Here the parameterisation is written $[2] = [2]$, with the first list being the columns to look in, the second list being the values to look for. The new data table is constructed by finding those tuples in the data table of C where the columns specified have the values specified. Then the columns with fixed values are discarded, and any repeated rows which result are also discarded.

3.5 The Compose Production Rule

The compose production rule covers both the composition of functions, and the specialisation of concepts, construction methods which are ubiquitous in every mathematical domain. HR can compose a concept C with a partner P by adding the predicates of P to those of C, effectively overlapping the concepts. For example, given two function concepts:

$$[n, x] : x = f(n) \quad \text{and} \quad [n, y] : y = g(n)$$

the construction can be function composition:

$$[n, x, y] : x = f(n) \wedge y = g(x) \text{ or}$$
$$[n, x, y] : x = g(n) \wedge y = f(x)$$

in which case the arity of the concept increases. It can also be a specialisation step:

$$[n, x] : x = f(n) \wedge x = g(n)$$

[the n are those special ones for which $f(n) = g(n)$]. The parameterisations are sets of numbers which specify how the predicates of P are to be overlapped with those of C. For example, if the parameterisation was $[1, 0, 2]$, this specifies that a new concept with predicates over three variables is to be constructed. The new concept will have the predicates of C. It will also have the predicates from P, but the variables will have changed - the first variable will be the first from the new concept, and the second variable will be the third variable from the new concept (as the number 2 appears in the *third* column of the parameters).

To build a data table for the new concept, HR runs through the data table for C and extracts any tuples, t, which overlap in the prescribed way with a tuple, s, from the data table of P. Then, for every pair $\langle s, t \rangle$, a tuple for the new data table is constructed by overlapping the tuples. For example, if the parameters were $[0, 1, 2]$, then we require a tuple $s = [s_1, s_2]$ and a tuple $t = [t_1, t_2]$ for which $t_1 = s_2$. If such a pair is found, the tuple $[s_1, s_2, t_2]$ is added to the new data table.

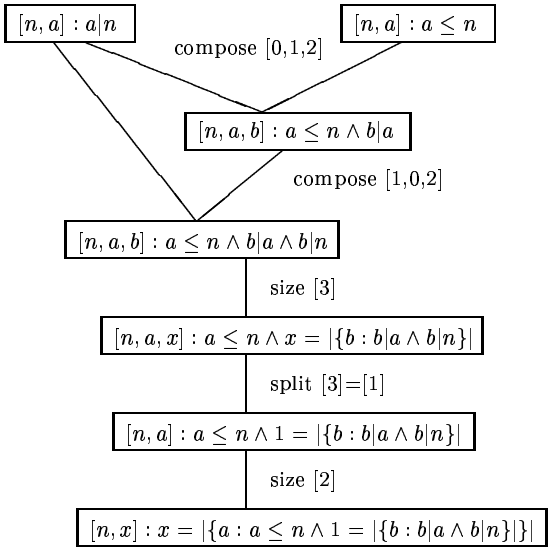

Figure 1. Construction of the ϕ function

3.6 Example Construction

Figure 1 details the construction of the number theory ϕ function which counts the number of integers less than or equal to n which are coprime to it (two integers are coprime if the only divisor they share is 1). This construction only requires 2 initial concepts (divisors and $\leq$), and 3 production rules (compose, size and split). This demonstrates that complicated concepts are reachable from fundamental concepts using just a few production rules. We define the **complexity** of a concept to be the number of concepts in its construction history. The complexity of the ϕ function is therefore 7. Often to impose a depth limit on the search we set a complexity limit of around 10.

4. A Forward Looking Mechanism

As each new concept can conceivably be combined with any other, and each production rule has a set of parameterisations, HR's searches lead to a combinatorial explosion. Asking HR to identify a concept of complexity 7 or 8 means searching a space which is too large to cover in a reasonable time. As discussed in Colton et al. (in pressb), HR usually works by choosing the best concept to use in the next theory formation step. HR has eight measures to decide which concept to use and the heuristic search is intended to increase the yield of interesting concepts.

We tried this approach for concept learning, using the invariance and discrimination measures described in Colton et al. (1999). These measure how close the categorisation produced by a new concept is to the categorisation produced by the goal concept. For example, suppose the goal concept is prime numbers, with this categorisation: [1,4,6,...],[2,3,5,...]. Concepts producing similar categorisations will score highly, and hence will be used first in the heuristic search. Unfortunately, this actually made matters worse. We noted a drawback which is common in heuristic searches - often to get to the concepts of real interest, it is necessary to go via dull concepts the search is designed to avoid. That is, certain concepts necessary to the construction of the goal concept may produce very different categorisations to that of the goal concept and be ignored by the search mechanism.

We therefore adopted another way to use the production rules - an exhaustive search enhanced by a mechanism for examination of new concepts in order to suggest which production rules (if any) to apply to them. Therefore, rather than identifying the present concepts which are best, the mechanism identifies which future concepts are best. For example, if trying to identify the prime numbers, 2, 3, 5, etc. given the initial concept of divisors of an integer, we would hope HR would notice that there were exactly two divisors for $2, 3, 5$ and so on. We would also want it to notice that the non-examples did not have this property, and suggest how to capitalise upon this. Similarly, if trying to identify the integers $1, 4, 6, 8, 9, 10, 11, 14, \ldots$, as soon as it invented prime numbers, we would want it to notice that the sequence members have no prime digits, and to suggest we compose the new concept of primes with the old concept of digits.

To do this, every new concept, C, is passed through each production rule, R, which searches for a pattern. If a pattern is found, a theory formation step stating that R should be used with C is added to the top of the agenda. For each production rule we have balanced the need to spot different patterns with the need to do this efficiently - looking too hard slows the search considerably. The size production rule, for instance, looks for 1 pattern: where the number of tuples in the data table is the same for every positive example. As soon as the pattern is broken, it stops looking, which improves efficiency. Only if the pattern is true for **all** the positive examples, will it look at the negative examples, in the hope that the pattern is not true for them. Only if no negative example has the pattern will the appropriate step be added to the agenda.

Some production rules effectively look two or even three steps ahead. The match production rule finds tuples (a, b, c) for which $b = c$. It can spot four patterns in this way: for every positive example, (i) all tuples have the property (ii) at least one tuple has the property (iii) the same number of tuples have the property and (iv) no tuples have the property. If any of these patterns are found, they not only suggest the use of the match production rule, but also the use of another production rule after that. For example, if no tuples are found, this suggests performing a match step followed by a negate step, and the agenda is updated accordingly. Similarly, if the same number of tuples with the property was found for each positive example, this would suggest performing a match step followed by a size step, followed by a split step. Hence in certain cases, this mechanism looks three steps ahead, allowing deeper access to the search space.

The compose production rule is treated differently, as patterns for this are sought by relating the newly formed concept to a partner concept. For efficiency reasons, HR looks at every possible partner, but only looks for a small number of common patterns. In particular, it notices that (i) the output of two functions are the same for each positive example [e.g., that the number of divisors equals the number of digits] (ii) that all the positive examples have the property of the concept and its partner [e.g., that an integer is both prime and odd] and (iii) that a set of subobjects (or the output from a function) has a property for each positive example [e.g., that each digit is prime]. The look-ahead functionality of the compose rule is very effective because HR often takes a long time to combine concepts when performing an exhaustive search.

5. Integer Sequence Results

We have initially concentrated on HR's ability at integer sequence extrapolation, due to the availability of the Encyclopedia to suggest sequences to learn. HR has so far identified 125 sequences from the Encyclopedia and 20 of the more well known ones are given

Table 2. Definitions for 20 well known number types

NUMBER TYPE	SEQUENCE	HR'S DEFINITION
balanced	$2, 9, 10, 12, ..$	$f(n) = g(n)$
composite	$4, 6, 8, 9, ..$	$\tau(n) \neq 1$ or 2
even	$2, 4, 6, 8, ..$	divisible by 2
even-square	$4, 16, 36, 64, ..$	even and square
evil	$3, 5, 6, 9, ..$	$f(n)$ is even
non-square	$2, 3, 5, 6, ..$	not square
odd	$1, 3, 5, 7, ..$	not divisible by 2
odd-prime	$3, 5, 7, 11, ..$	odd and prime
odd-square	$1, 9, 25, 49, ..$	odd and square
odious	$1, 2, 4, 7, ..$	$f(n)$ is odd
power-of-two	$1, 2, 4, 8, ..$	1 odd divisor
prime	$2, 3, 5, 7, ..$	2 divisors
prime-digits	$2, 3, 5, 7, ..$	all digits are prime
prime-power	$2, 3, 4, 5, ..$	1 prime factor
prime-squared	$4, 9, 25, 49, ..$	square of a prime
repdigit	$1, 2, 3, 4, ..$	only 1 distinct digit
repunit	$1, 11, 111, ..$	repdigit, 1 is a digit
semi-prime	$4, 6, 9, 10, ..$	2 prime factors
square	$1, 4, 9, 16, ..$	of the form $n \times n$
square-free	$2, 3, 5, 6, ..$	1 square divisor

in table 2, with the (paraphrased) definitions that HR produced for them. We let $f(n)$ be the number of 1's in the binary representation of n and $g(n)$ be the number of 0's. In most cases, HR finds an instantly recognisable, correct definition. In some cases HR finds alternative definitions, e.g., it notices that powers of two are the only integers with exactly one odd divisor.

The average time to learn one of these 20 concepts is 373.05 seconds with an exhaustive search, reducing to just 3.65 seconds when the forward looking mechanism is employed. The most striking examples of the efficiency gain are the cases where two simple concepts are combined into a more complicated one. For example, odious numbers are those with an odd number of 1's in their binary representation, e.g., 25 is odious because it is written 10011 in binary, with three 1's. HR cannot learn this until it has invented both the concept of odd numbers and the function which counts the number of ones in the binary representation of an integer. These were the 26th and 54th concepts produced respectively. When concept 54 was introduced, the forward looking mechanism determined that it should compose with concept 26 and this led to the solution after only 7 seconds. Without the heuristic, HR took 90 minutes to get around to composing concepts 26 and 54.

As well as learning well known sequences, we identified sequences missing from the Encyclopedia of Integer Sequences. Given any integers a, b, c and d such that $0 < a < b < c < d < 10$, the Encyclopedia has a sequence beginning a, b, c, d, with just two exceptions: there are no sequences starting with $4, 5, 6, 9$ or $4, 5, 7, 9$. We used HR to extrapolate these. The sequence $4, 5, 7, 9$ was very easy, HR simply invented the concept of primes + 2. The sequence $4, 5, 6, 9$ was more difficult. The solution HR found uses the binary representation of an integer, n, to write it as $n = 2^{j_1} + 2^{j_2} + \ldots + 2^{j_k}$. For example $11 = 2^0 + 2^1 + 2^3$. HR first used the exists production rule to define the set: $b(n) = \{j_1, j_2, \ldots, j_k\}$, e.g., $b(11) = \{0, 1, 3\}$. Then it composed this with the concept of divisors and defined those divisors which appear in $b(n)$. For example, $b(6) = \{1, 2\}$ so divisors 1 and 2 of 6 appear in $b(6)$. Finally, HR negated this concept, looking at divisors which do not appear in $b(n)$, and it spotted a pattern for the integers $4, 5, 6$ and 9: they have exactly 2 divisors which do not appear in $b(n)$. ie. $4, 5, 6, 9$ are the first four integers, n, for which:

$$|\{a : a|n \wedge a \notin b(n)\}| = 2.$$

This sequence continues: 4, 5, 6, 9, 13, 14, 15, 17 and we see that HR has intelligently extrapolated $4, 5, 6, 9$. However, this sequence seems to have little value other than providing an answer to our question.

6. Related Work

The AM program (Davis & Lenat, 1982) worked in number theory, re-inventing concepts like prime numbers and making conjectures such as Goldbach's conjecture. Starting with 115 concepts, AM applied one of 242 heuristics to determine which task to do next - necessary as it often had over 2000 tasks on the agenda. Some heuristics were very specialised, enabling AM to reach particular concepts and sometimes the user guided AM. HR's theory formation is much simpler: it starts with only a few initial concepts, uses only seven construction techniques and has only eight measures of interestingness. AM was never applied to machine learning tasks such as identifying particular concepts.

The Graffiti program, (Fajtlowicz, 1988), makes conjectures in graph theory stating that one summation of numerical invariants is always less than another summation. Its concept formation is limited to summing invariants, and it is not used to identify concepts. The simply stated but difficult conjectures have efficiency applications and have been settled by many notable mathematicians. In (Colton et al., in pressb), we compare HR, AM, Graffiti and similar discovery systems.

6.1 Integer Sequence Extrapolation

The Encyclopedia of Integer Sequences comprises around 54,000 sequences collected by Neil Sloane, with contributions from many mathematicians. The user supplies the first few terms of a sequence and the Encyclopedia is searched until a sequence is found which contains the given terms. There is also an email server called the superseeker which works much

harder to find a match for a given sequence. Superseeker transforms the input sequence and searches for a match for the transformed sequences. Such transformations include taking the difference between successive terms and more complex manipulations such as the Boustrophedon transformation by Millar et al. (1997). Superseeker's transformations are very good at identifying sequences related to one already in the Encyclopedia. However, due to the size of the database, it has limited ability to relate two sequences. For instance, even though there are many sequences about the digits of an integer, and the sequence of prime numbers is fundamental, the superseeker cannot determine the nature of these numbers: 1,4,6,8,9,10,11,14, which are simply those with no prime digits.

The SeekWhence program by Hofstadter (1995), was designed from a cognitive science perspective to extrapolate integer sequences. Early versions determined the nature of an integer sequence by performing some mathematical transformations, such as taking differences between terms, and some general pattern recognition transformations, such as looking at every second term in the sequence, or identifying repeating clusters. For example, to extrapolate the sequence $1, 4, 9, 16, \ldots$, SeekWhence would first transform it by taking the difference between successive terms to give: $3, 5, 7, \ldots$ It would then recognise this as a sequence in its database, ie. odd numbers. As it knew how to extrapolate odd numbers, it could derive a way to extrapolate the original sequence. The project originally aimed to outperform mathematical approaches to sequence extrapolation, such as those described by Persson (1966). However, they became more interested in what Hofstadter calls 'Haiku' sequences, which are independent of the context of mathematics, and only require general pattern recognition rules to extrapolate. In this way, SeekWhence could also extrapolate non-mathematical sequences, in particular melodies.

HR occupies the middle ground between the Encyclopedia and SeekWhence. The production rules are designed to be applicable to many domains, so it does not use the domain specific transformations of the Encyclopedia. Nor does it use general pattern finding templates and heuristics like SeekWhence. This is because HR is primarily a machine discovery program employed to invent new concepts - the patterns of SeekWhence are so general they would produce a plethora of concepts, many of which would be of little interest to mathematicians. By identifying certain general properties of mathematical concepts, we have given HR a mathematical pattern generating ability which it can use to invent new concepts in an attempt to find a definition for a given set of examples.

6.2 A Qualitative Comparison with Progol

The Progol program (Muggleton, 1995) uses inductive logic programming (ILP) to learn a concept given a set of predicates as background knowledge and a set of positive and negative examples for the concept. There is a striking similarity between the concepts Progol and HR can reach. Firstly, Progol uses inverse resolution to invent predicates which could have been resolved to produce the background and example predicates. This produces concepts with conjunctions of predicates, predicates with repeated variables, and conjunctions of predicates which contain the same variable. We have found that this covers the concepts that HR can produce with its compose, match and exists production rules. For example, HR produces this definition for square numbers: integers n such that $\exists\ a$ s.t. $a \times a = n$, and Progol produces this definition:

```
square(N) :- integer(M), multiply(N,M,M).
```

Secondly, the user is allowed to set mode declarations detailing where background predicates can appear in the invented predicates. Mode declarations also specify whether variables become instantiated and whether negation of predicates is allowed. The ability to instantiate variables corresponds exactly with HR's split production rule, and the ability to negate predicates corresponds with the negate rule. A combination of negated and existentially quantified predicates corresponds to concepts produced by HR's forall production rule. For example HR produces the definition for even numbers as being divisible by two. Given the background predicate of divisors and allowed to instantiate variables, Progol produces this definition:

```
even(N) :- divides(N,2).
```

Finally, we found that if we supply two additional predicates as background knowledge from set theory, namely the standard Prolog predicates of `setof` and `length`, Progol can cover concepts produced by the size production rule. For example, HR defines prime numbers as having exactly two divisors, and Progol produces this equivalent definition:

```
prime(N) :- setof(M,divides(N,M),L),
            length(L,2).
```

Therefore, for each of HR's production rules, we have found a way for Progol to produce concepts of a similar nature. Interestingly, to cover all the production rules requires three different aspects of Progol's functionality. Only one of HR's production rules corresponds to additional background knowledge, which adds to our claim that the production rules are very general. We are currently undertaking a quantitative assessment of HR and Progol to enable us to better compare and contrast issues such as coverage, efficiency and control.

7. Further Work and Conclusions

Our approach to the identification of mathematical concepts is specific to mathematics, but uses no information specific to any domain and provides a way to identify types of graphs, types of groups and types of numbers with very little modification to the program. It is likely that the best approach to identifying mathematical concepts will combine programs with:

- A large database, e.g., the Encyclopedia.
- Domain specific transformations, e.g., Superseeker.
- General pattern recognition, e.g., SeekWhence.
- Inventive abilities, e.g., Progol, HR.

HR starts with the fundamental concepts of a domain, modelling the way in which a concept can be learned completely from scratch. HR would benefit from a knowledge base of interesting concepts in a domain. Using concepts like prime numbers as the basis for theory formation rather than more fundamental concepts like divisors, it could progress further into the search space. It would also benefit from some domain specific transformations, and we have begun implementing some based on those used by superseeker.

We can also look at the concepts which Progol can but HR cannot learn, and suggest new production rules to fill the gap. Progol can generate recursive definitions, such as the factorial function. We plan a 'path' production rule to enable HR to construct concepts with recursive definitions. Also, Progol is able to produce definitions with disjunction of predicates, such as integers which are prime *or* square. We have so far avoided a 'disjunct' rule, worried that the theories produced will contain too many dull disjunctions, but we plan to implement it for machine learning purposes.

As discussed by Langley and Michalski (1986), there is much overlap between machine learning and machine discovery, with machine learning tools discovering new results in science, e.g, ILP in biology. We have discussed the reverse problem here: how to apply a discovery program to machine learning problems. Presenting HR's production rules as a tool for machine learning, we have provided some justification for each based on observations from the mathematical literature, and detailed how each forms concepts. We have shown that the search space defined by the rules can reach many complicated concepts in number theory such as the ϕ-function and that our forward looking mechanism greatly improves efficiency.

The project to apply HR to machine learning tasks is far from complete. A qualitative comparison of HR and Progol has highlighted that HR's production rules correspond with various aspects of Progol, showing that HR can theoretically cover many (but not all) of the concepts Progol can learn. We are presently undertaking a quantitative comparison of the systems. In Colton et al. (in pressa) we have shown that HR can invent interesting integer sequences, in effect posing sequence extrapolation problems. We hope to have shown here that it can also solve them.

Acknowledgements

This work is supported by EPSRC grant GR/M98012. We would like to thank the anonymous reviewers for their very helpful comments, and Stephen Muggleton for many in-depth discussions about ILP and HR.

References

Colton, S., Bundy, A., & Walsh, T. (1999). HR: Automatic concept formation in pure mathematics. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (pp. 786–791).

Colton, S., Bundy, A., & Walsh, T. (in pressa). Automatic invention of integer sequences. *Proceedings of the Seventeenth National Conference on Artificial Intelligence.*

Colton, S., Bundy, A., & Walsh, T. (in pressb). On the notion of interestingness in automated mathematical discovery. *International Journal of Human Computer Studies.*

Davis, R., & Lenat, D. (1982). *Knowledge-based systems in artificial intelligence.* New York: McGraw-Hill.

Fajtlowicz, S. (1988). On conjectures of Graffiti. *Discrete Mathematics, 72,* 113–118.

Hofstadter, D. (1995). *Fluid concepts and creative analogies.* New York: Basic Books.

Langley, P., & Michalski, R. (1986). Machine learning and discovery. *Machine Learning, 1,* 363–366.

Millar, J., Sloane, N., & Young, N. (1997). A new operation on sequences: the Boustrophedon transform. *Journal of Combinatorial Theory, 17A,* 44–54.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing, 13,* 245–286.

Persson, S. (1966). *Some sequence extrapolating programs: A study of representation and modelling in inquiring systems* (Technical Report STAN-CS-66-050). Department of Computer Science, Stanford University, Stanford, CA.