The HR3 Discovery System: Design Decisions and Implementation Details

Simon Colton, Ramin Ramezani and Maria Teresa Llano[1]

Abstract. Automated Theory Formation is a hybrid AI technique which has been implemented in two scientific discovery systems, HR1 and HR2, both of which have been used successfully in various applications. We describe here the latest iteration in the HR series, in terms of the lessons learned from the successes and failures of the previous versions, and how these lessons have informed our design choices and the implementation details of the new version. We also present two case studies: a synthetic domain mirroring an aspect of medical diagnosis, and invariant discovery in formal methods. In each case, we compare HR3 with HR2 to highlight various improvements in the new version.

1 Introduction

As argued in [13], while this is a broad categorisation, there have been two major paradigms in Artificial Intelligence research: *problem solving* and *artefact generation*. In the former, an intelligent task to automate is interpreted as a series of problems to be solved; in the latter, an intelligent task is interpreted as a series of artefacts of value to be generated. As an illustrative example, there are automated theorem provers which act as decision procedures and say simply "yes" or "no" to questions about the truth/falsehood of a given conjecture (problem solving), and there are some which produce proofs which can be interrogated and understood (artefact generation). While many systems for scientific discovery have been developed within the problem solving paradigm, and such applications do indeed require much problem solving, we believe that scientific hypotheses, examples and proofs/evidence are artefacts of value, and hence the latter paradigm would appear to be a more natural setting. In the artefact generation paradigm, it is important to write computational systems that *produce* rather than *solve*, and this can mean keeping data that would be discarded in a problem solving setting and/or searching parts of a space which could be avoided when all that is required is a single answer. While this has clear computational disadvantages, there are advantages to artefact generation approaches, which we highlight at various stages below.

Firmly within the artefact generation paradigm, we have developed a series of discovery systems which perform *Automated Theory Formation* (ATF), as described in section 1.1 below. Largely applied to mathematical invention tasks, but also with applications in other domains, our HR1 and HR2 ATF systems have been used with much success. However, in recent years, various limitations related to the design and implementation of HR2 have become apparent. In particular, the speed at which it operates and its memory consumption have held it back, and we have been forced to use other systems such as the WEKA machine learning suite [15] and the Progol [24] Inductive Logic Programming (ILP) system for applications where HR2 would naturally fit, due to the size of the search space encountered.

Given these failures, we have gone back to the drawing board and built a new version, HR3, from scratch. Memory footprint and search efficiency have been paramount considerations in the new design. In addition, in comparing HR2 with other systems, notably ILP implementations like Progol, as in [5], we have identified the unique aspects of the ATF approach, and sought to preserve them in the new version. The design decisions are described in section 2, and details of how these have influenced the implementation of HR3 are given in section 3. We describe a novel way for discarding uninteresting conjectures through the use of random theories in section 4. In sections 5 and 6, we present two case studies with a scientific discovery flavour, to compare and contrast HR3 with its predecessor. We conclude in section 7 and discuss some avenues for future development and deployment of the HR3 software.

1.1 Background

Automated Theory Formation was introduced as a hybrid AI technique for discovery tasks in [8] and then through a series of implementations and papers, leading to this one. While 'Automated Theory Formation' is a rather grandiose and perhaps too generic title for this approach, we chose it to reflect that everything one would expect as bare necessities in a mathematical theory (examples, concepts, conjecture, theorems and proofs) are produced with the ATF approach. To form such theories, the HR2 system starts with minimal background knowledge as would be given to a machine learning system, in Prolog notation, or just the axioms of a mathematical theory in first order logic. Using a set of production rules, it takes old concepts and generates new ones from them, using logical constructions, including the usage of universal and existential quantification, composition, disjunction, etc., and/or using mathematical constructions, such as counting, the usage of inequalities and statistical summarisation methods such as averaging, finding maxima, etc.

The concepts categorise and describe the given examples and the system can be seen as performing unsupervised learning. From the concepts, HR2 finds non-existence, implication and equivalence conjectures empirically by noticing patterns in the data, e.g., finding that the success set of one concept is a subset of that of another, which prompts the formation of an implication conjecture. In this sense, the system is best described as data mining conjectures, or as a descriptive machine learning system. Finally, when axioms are provided in the first order logic syntax of a theorem prover such as Otter [21] or model generator such as MACE [22], then HR2 uses these to attempt to prove the truth of the conjecture, or find a counterexample,

[1] Computational Creativity Group, Department of Computing, Goldsmiths, University of London `ccg.doc.gold.ac.uk`

which is added to the theory. The ATF routine is driven by measures of interestingness [11] which guide a heuristic search by choosing the next concepts to develop with the production rules, and as such are somewhat similar to the mode declarations used in ILP systems.

It is beyond the scope of this paper to give full details of the ATF project, so we highlight the three milestone publications which have appeared over the last decade. Firstly, in [7], we describe the theory behind ATF and its implementation in the HR1 software, with applications to mathematical discovery. Secondly, in [12], the HR2 system is projected as an Inductive Logic Programming system, and a summary of the many applications to which we applied it is given. Such applications include the reformulation of constraint satisfaction problems [4], generating novel algebraic classification theorems [28] and the automatic invention of integer sequences [9]. Thirdly, in [25], certain more sophisticated approaches to theory formation are presented, drawing on theories such as Global Workspace Architectures [1] and Lakatos's philosophy of mathematics [17].

2 Design Considerations

HR2 employs *production rules* to turn old concepts into new ones and empirical conjecture making techniques to find relationships between the concepts. Used in various modes, it can introduce new constants produced by third party model generation, constraint solving and/or computer algebra systems, and new proofs via third party Automated Theorem Provers (ATPs). The generation of a variety of types of output, and variety within each type has been a feature of HR2, and there have been many applications where we had no idea in advance whether an example, concept, conjecture or proof would be the most interesting thing about a domain investigated by HR2. More than any other, this feature has enabled us to present HR2 as a creative system in a Computational Creativity setting [13], because of the surprising and high-quality nature of the artefacts produced. Hence our first main design consideration for HR3 was to maintain and improve upon this diversity of output artefact. To this end, we have implemented production rules able to deal with floating point numbers (which HR2 never did well) and textual data such as tweets, and to enable HR3 to read input in a variety of formats, including Prolog and CSV files, ontologies and Java code.

The variety of output is clearly enabled by HR2's reliance on third party software. However, the variety of concepts that HR produces is very much restricted by the requirements that these systems put on the nature of the concepts that HR2 can produce. In all the applications with ATPs, for instance, HR2 was only allowed to use its subset of production rules which generate concepts expressible in first order logic. While it has been presented as an ILP system in [12], HR2 was never intended to be dependent on a particular logical formalism. It achieved this by having two separate processes: one for generating the data supporting a concept, and another for generating a definition (or definitions) for a concept. While this has been a little difficult to maintain, it has allowed us to implement production rules that invent concepts with only sketchy definitions that a mathematician might understand – as the definition is not linked to the construction of a success set for a concept, we have this freedom. This has enabled us on many occasions to implement bespoke production rules and concept evaluation techniques, enabling HR2 to be applied to tasks where a logical representation would impose restrictions. Hence, our second main design consideration was to maintain this lack of reliance on logical formalisms in HR3, and to increase independence from third party systems which are reliant on first order and other logics. To this end, we have implemented an approach to redundancy reduction which removes one reliance on ATP systems, as described in section 4 and demonstrated in the case studies below.

The majority of HR2's processing is done to no avail: it produces concept definitions that no-one ever reads, makes conjectures that no-one will investigate, calls third party theorem provers and model generators to determine the truth of conjectures that no-one is interested in, and constantly checks and discards theory material, which takes an inordinate amount of time. We found that the extra processing often ground HR2 to a halt, and so we took the decision to implement an *on-demand* model for HR3. In particular, at theory formation time, HR3 performs the barest amount of work in order to form a theory, and only after, when the user chooses which of the material to look at, is extra processing performed in order to present the concepts and conjectures in a coherent way. We realised that HR3 does not need to produce definitions during theory formation, and while it records information for non-existence and equivalence conjectures, it doesn't formulate these explicitly either, nor does it look for implication conjectures at all. In addition to slowing theory formation down, the additional material produced added a lot to the memory footprint, and we would often find that HR2 was pushing memory boundaries when it started with a large background theory and produced many concepts and conjectures. For this reason, with HR3, we decided to keep the memory footprint as low as possible. How this is achieved, and how the bare details of a theory are expanded on demand are described in section 3 below.

HR2 has a graphical user interface with more than 300 widgets on it, which help users to set up the background knowledge and search and reasoning settings for a session, and to interrogate the results both during and after theory formation. This has been particularly difficult to maintain. Moreover, we often found that, after a theory had been formed, the inspection functionality was not sufficient, as there was an aspect of the theory that was not accessible, or a calculation that we didn't know in advance we wanted to perform. Given that the theory formation sessions could take many hours, this was often frustrating. Hence, we implemented text boxes into which users could write Java code, and then a hand-crafted interpreter would use this at run-time to generate reports. This was not easy to maintain, and has been superseded by class-loading technologies in Java.

While we plan to build simple interfaces to HR3, in particular to enable it to act as a web-service, we have decided that the primary interface will be Java programming. Hence HR3 will be made available as an API, rather than stand-alone software. This will acknowledge the fact that the likely power-users of HR3 will be programmers, while casual users will use the web interface, and is in line with modern trends in software deployment. However, there is another important reason to make the primary interface Java code: to recognise and develop HR3 as an automated programming system. We see software writing software (creatively) as a major future direction in Computational Creativity research. Moreover, machine learning programs perform automated programming, which is most obvious in ILP and evolutionary programming approaches, but true in general. HR3 is being developed to generalise past the restrictions the machine learning paradigm imposes via logical and other formalisms on concept representation, and the tasks systems are applied to. Automated Theory Formation as per HR3 really involves the production of (i) sets of algorithmic procedures from which data and definitions can be generated if needed, (ii) relationships between the processing done by the procedures on different types of data, and (iii) proofs (or empirical demonstrations) of the likely truth of the relationships. Hence, when describing the implementation details below, we often talk about concepts, constructions and procedures.

3 Implementation Details

As with HR2, we split the timeline for working with HR3 into *theory formation time*, where it is working uninterrupted on forming concepts and conjectures, and *theory interrogation time*, where various routines are run to generate more information about what has been produced. Given that the primary interface to HR3 is writing Java programs, the programmer is at liberty to partially form a theory, then interrogate it, then form more of the theory, perhaps dependent on the results of the intermediate interrogation, etc. HR3 generates and records remarkably little information at theory formation time, and the information it does keep is very slimline: only lists of integers. Background constants and predicate names are stored as strings, but the relationships expressed on the *tuples* of constants by the predicates are stored as integer lists. Then, when new a new concept is added to the theory, HR3 records (i) a list of integers for each concept, which point to tuples of objects, which are themselves integer lists pointing to Strings – this acts as the datatable describing the success set of the concept, and (ii) a set of integer lists for each concept, representing the different constructions which led to the datatable of the concept (hence storing information about equivalence conjectures). Whenever a newly formed concept has an empty set of supporting tuples, the construction leading to this is recorded as a list of integers, and nothing else. To reduce the memory footprint and increase processing efficiency, we represent integer lists as TIntArrays from the Trove API for high performance Java collections, available at: `http://trove.starlight-systems.com`.

For example, suppose we start with background knowledge about animals, (a toy dataset that often ships with ILP systems), such as:

```
animal(dog).animal(herring).integer(0).integer(4).
legs(dog,4).legs(herring,0).milk(dog).
```

When reading this, HR3 would assign constants dog, herring, 0 and 4 the integers 0, 1, 2 and 3. Then tuples (dog), (herring), (0), (4), (dog,4), (herring,0), (dog) would be mapped to (0), (1), (2), (3), (0,3), (1,2), (0), which would in turn be assigned tuple numbers 0, 1, 2, 3, 4, 5 and 0. Finally, the datatable for the concept of 'animal' would be recorded as the list [0,1], as it contains the first and second tuples, the datatable for 'integer' would be [2,3], the datatable for 'legs' would be [4,5] and the datatable for 'milk' would be [6].

HR3's production rules are very similar to those implemented in HR2. Their main task at theory formation time is to take one or two datatables of individual concepts and output a new datatable. Storing datatables for concepts as a list of integers is troublesome for some production rules, as it requires unpacking and manipulating of the tuples that the integers point to, and then further unpacking the tuples. However, for some of the most used production rules, this representation makes processing very efficient. In particular, for the *conjunction* production rule, given two old datatables, the new one is simply the intersection of the two lists of integers. This rule is unique in always being used in theory formation, and, being a binary production rule, it is used for the majority of theory formation steps. The *disjunction* binary production rule also has a very efficient implementation which simply finds the union of two lists of integers, and *negation* subtracts one list from another, hence is similarly fast. Unlike HR2, where production rules are applied piecemeal, HR3 applies production rules in *blocks*. This has further efficiency gains, as sub-blocks of steps can be distributed over multiple threads (see later). Checking whether a newly generated datatable is the same as a previous one is important, as making equivalence conjectures rather than new concepts reduces search significantly. We tested various methods for this, and found that a double-hashing approach, where hashtables are stored as entries in a hashtable, was the most efficient.

In addition to generating a new datatable for a new concept, each production rule also generates a list of integers which contains information about how the datatable was constructed which would be sufficient to repeat the process. For instance, the *conjunction*, *negation* and *disjunction* production rules record only the numbers of the concepts whose datatables were combined. The *existential* and *count* rules record which concept number they were applied to, and the column in the tuples to which existential quantification and counting was applied respectively. Whenever a newly formed concept is produced which has the same datatable as a previous concept, the construction is added to a *construction set* for the latter. Each *construction* contains by default a number pointing to the production rule used, and they can be used for re-building concepts over random data, as explained in section 4.

At theory interrogation time, prompted by the user, an individual construction for a concept can be expanded into a set of *procedures* which are nested integer lists. For instance, supposing the *conjunction* production rule was represented by the number 3, then the construction [3,17,20] specifies that *conjunction* was applied to concepts 17 and 20. We can therefore take any construction from the construction set of concept 17 and substitute it in this construction, e.g., [3,[2,14],20], and do likewise for concept 20, producing, for instance: [3,[2,14],[3,10,11]]. This could be further expanded, continuing until a cycle occurs or optionally a background concept is reached, in which case it might be useful to not substitute it with any of its alternative constructions.

Users can set an *expansion depth limit* to the number of substitutions that can be made, which limits the number of procedures generated for any construction. This is done by recording the depth of expansion and only allowing the first construction for each concept to be used when the depth limit is passed. Often, the number of different procedures for a concept can be enormous, and this is exacerbated when two concepts appear in a conjecture, so a depth limit is very useful, as we see in the second case study below. We say that a construction has been expanded to *completion* when all the procedures have been produced from it, with no depth limit. Each procedure represents fully a different algorithm for manipulating the background knowledge to produce the data for the concept from which it was derived. HR3 also uses procedures to generate definitions for a concept. Working iteratively from the original predicate definitions given in the background theory, each production rule has bespoke code able to change the definition(s) that are given to it, with the input definitions to each production rule dictated by the procedure.

We are embracing parallelism fully in the new version of HR. In particular, we have implemented a load balancing mechanism where theory formation steps can be distributed across the available threads in a machine. We have experimented with two setups for this: (a) a single theory is maintained and each theory formation thread contributes to this in a synchronised way, or (b) each thread gets a completely independent copy of the theory, then adds material to this, which is collated at the end, in a synchronised way. We have found that method (b) is most effective when the domain is sparse in terms of the number of different concepts in it. As the threads are independent in terms of data, a nearly-linear speed up in run time is achieved. However, when each thread finds many new concepts, there is a bottleneck at the end with method (b), as concepts from each thread have to be compared to those from all the other threads in a linear fashion before being added to the theory, to avoid repetitions. In these circumstances, distribution method (a) can be faster.

System	Threads	Steps	Concepts	Memory (Gb)	Time (ms)	Steps/s
HR2	1	1000	248	0.31	3000	333
HR2	1	10000	755	1.13	11000	909
HR2	1	50000	1846	2.56	56000	893
HR2	1	100000	3540	2.73	151000	662
HR3	1	1114	57	0.00	49	48000
HR3	1	2315	210	0.00	57	68000
HR3	1	42074	2875	0.01	152	336000
HR3	1	991143	12230	0.13	485	2043000
HR3	1	75629333	16392	0.05	18413	4107000
HR3	2	75629333	16392	0.33	10008	7556000
HR3	4	75629333	16392	0.60	6519	11601000
HR3	8	75629333	16392	1.66	5549	13693000

Table 1. Efficiency results for HR2 and HR3 on a 2.6Ghz machine.

3.1 An Efficiency Comparison

As mentioned previously, time and memory efficiency was a major concern with HR2 and we have optimised HR3 to be much better in these respects. To compare the systems, we ran both to form a theory from the animals dataset, using the *compose/conjunction*, *split/instantiation* and *negate/negation* production rules. For HR2, we ran it for a particular number of theory formation steps, whereas we ran HR3 for particular sequences of production rule blocks. We imposed no constraints on the theory formation in either case, and told both systems to keep information about all concepts, equivalence conjectures and non-existence conjectures it encountered. For HR3, we further experimented with 1, 2, 4 and 8 threads employed during theory formation using thread distribution setup (b) above. The results of the experiments, where we compared the efficiencies and memory footprint of HR2 and HR3, are presented in table 1.

Looking at time efficiency, we see that HR2 suffers from a relatively large start-up cost, meaning that it performs at 333 steps per second at 1,000 steps, performs at best when the session is more substantial (raising to 909 steps/s), but its performance degrades as the theory it is dealing with grows in size (reducing to 662 steps/s). At its fastest (13.6m steps/s), HR3 is 15,000 times faster than HR2 at its fastest (909 steps/s). This is somewhat misleading, however, as HR3 gained from searching a larger, largely empty space (in terms of new concepts). Comparing (roughly) like for like in terms of the number of theory formation steps, we see that, at 1,000 steps, HR3 is around 100 times faster than HR2, while at 10,000 steps, HR3 is around 300 times faster. Note that efficiency gains recorded in the case studies below are much more modest, largely because of the small theory formation sessions required to solve the problems in each case. Using 8 threads, HR3 searches a space of 75.6 million concept constructions in 5.5 seconds. Again, while this is indicative of the raw speed of HR3, this is somewhat misleading, as in the animals domain, the concept space is very sparse, hence there was virtually no bottleneck waiting for each thread to offload its results sequentially.

Regarding memory efficiency, HR2 rapidly consumed gigabytes of memory, because it was generating information that would never be seen. In contrast, even with 12,230 concepts, 896,285 equivalences and 82,628 non-existence conjectures stored, HR3 only used 130Mb of memory. Note that we turned off the storing of equivalence and non-existence conjectures for the very large sessions, as we found that this caused memory issues for HR3. In these large sessions, we see that copying the theories for the multi-threaded approach imposed a memory cost, and we plan to look into reducing the memory footprint in these cases.

4 Tidying Theories using Random Data

A practical reality of working with an ATF system is an overwhelming volume of examples, concepts and conjectures produced, which can be so numerous (in the millions) that they have to be carefully pruned before presentation. HR3 makes three types of conjecture empirically: (i) $\nexists\ t$ s.t. $N(t)$, i.e., non-existence conjectures stating that, given the semantics of the domain, no tuple of constants t exists which satisfies the definition of concept N, or equivalently: when the construction process for concept N is performed, the result is an empty set, (ii) $\forall\ t\,(L(t) \rightarrow R(t))$, i.e., implication conjectures stating that, given the semantics of the domain, the definition of concept L implies the definition of concept R, or equivalently: when the construction process for concept L is performed, the resulting set of tuples is a subset of those acquired when the construction process for concept R is performed, and (iii) $\forall\ t\,(L(t) \leftrightarrow R(t))$, i.e., equivalence conjectures, explained similarly to implications.

One particularly annoying subset of material that needs to be found and discarded are tautological conjectures such as $((A \wedge B) \leftrightarrow (B \wedge A))$, etc. In HR2, we employed a variety of tactics to avoid outputting these kinds of results. These included (i) *forbidden paths*, which constrained the concept formation, e.g., HR2 would not invent the concept $(B \wedge A)$ if B appeared later in the theory than A, hence the above tautology would not happen (ii) in-built reasoning that worked with the definitions of concepts in conjectures to determine whether they were obviously true, and (iii) appealing to a third party automated theorem prover to show that a conjecture was true even without any axioms. Taken together, these approaches were fairly effective, but by no means perfect; difficult to maintain, as each new production rule required much additional coding to rule out the particular forms of tautologies that it introduced; and, above all, very time consuming. We have opted for a radically different approach with HR3 which is much faster, more reliable and requires no additional functionality in the production rules.

At its core, the approach requires the generation of a *random* counterpart to a given background theory. By that, we mean that the random counterpart has the same number of constants as the original background theory, the same number of predicates with the same arities, and the same number of data points supporting each predicate. For instance, in the animals dataset, there are 18 constants and 13 background concepts represented by 9 unary predicate with 18, 4, 3, 4, 4, 5, 8, 14 and 14 supporting singletons respectively, and 4 binary predicates with 18, 18, 18 and 22 supporting pairs of constants. The random counterpart matches this structure exactly, but fills in the singletons and pairs with randomly chosen constants. For instance, the data for the 'class' predicate, with 18 points such as `class(dog, mammal)`, `class(herring, fish)` which reflect reality, would be replaced in the random theory with a binary predicate *rclass* with 18 data points chosen randomly, such as `rclass(rconst10, rconst17)`, `rclass(rconst12, rconst16)`, etc.

Given a conjecture, C, empirically true of the original data, if C is a tautology to be removed, this means that the underlying semantics of the domain don't have to be taken into account to prove the truth of C. The constructive analogy is that: no matter what data tuples are given to begin with, the construction of the concepts related by the tautology will always lead to that relation being upheld. Random theories are constructed with only the syntax of the original theory used, and none of the semantics of the underlying data taken into account. Hence, a conjecture which is true about the original data can be tested by constructing the related concepts from the random data and seeing if the relation still holds. This is enabled by applying the procedures linked by a conjecture to the random data. If a conjecture still holds for random data, this will be for one of these reasons:

1. The relationship happened by random chance.

2. The relationship happened as an artefact of the construction processes leading to the related concepts, which had nothing to do with the underlying data.

3. Sparsity of data coupled with the construction processes producing highly generalised or specialised concepts mean that, while the conjecture is true of the random data, it is not true in the general case.

4. The relationship happened because the data which would break this conjecture has not been sampled due to a small random sample size used in constructing the random theory.

In practice, we generate multiple random theories as above, then test conjectures against them. If all the random theories support the conjecture, this provides much evidence that reason 2 above (corresponding to the conjecture being tautologous and not interesting) is correct. Under testing of this approach, we have observed that by increasing the number of random theories tested, the probability of reasons 1, 3 and 4 causing an empirically true relation being true in a random theory reduces greatly. Intuitively, the random theories approach asks the user to consider statements like the following: "The following conjecture was true in your original data, so it might say something of interest about your domain. However, when tested against 100 randomly generated theories, this conjecture was shown to be true in all cases. Do you still think it says something interesting about your data?" Under normal circumstances, it is highly likely that the cause of the relationship holding for the random datasets is reason 2 above: independent of any underlying semantics, the constructions performed by the production rules always relate data thus. We have found that this method always removes all tautologous conjectures, as they are always true regardless of the nature of the random theory.

However, in circumstances of sparse data and theory formation with much generalisation or specialisation, the approach can also remove some false positives, i.e., conjectures which are not true in the general case, are discarded because of reason 3 above. For instance, consider the conjecture $\forall t(A(t) \wedge B(t) \wedge C(t)) \leftrightarrow (A(t) \wedge B(t) \wedge D(t))$. Suppose that the number of supporting examples for concepts A and B was 5 each, out of a total of 20 tuples of the correct type. Then, if concepts A, B, C and D were constructed randomly, the likelihood of a tuple t being such that $(A(t) \wedge B(t))$ is true would be $0.25 \times 0.25 = 0.0625$, as the construction of A and B were independent events. Hence, we can expect only $20 \times 0.0625 = 1.25$ tuples of the 20 possible to satisfy $(A(t) \wedge B(t))$. So, we expect at most 1 tuple, T, to be true of both the left hand and right hand side of the conjecture, given the constraint of satisfying $(A(t) \wedge B(t))$ imposed on both sides. Finally, suppose concepts C and D have 10 supporting examples out of the 20 possible. Hence, the likelihood of T supporting the left hand side is 0.5 and supporting the right hand side is 0.5 also. So, the probability of the conjecture $\forall\, t\,(A(t) \wedge B(t) \wedge C(t)) \leftrightarrow (A(t) \wedge B(t) \wedge D(t))$ being true is calculated at $2 \times (0.5 \times 0.5) = 0.5$.

We see that, given the sparsity of data for the domain, this conjecture is likely to be true of half the randomly generated theories it is tested against, while clearly not a tautology. Hence it would seem unwise to discard it based on the evidence of one random theory. To mitigate the risk of throwing away non-tautologous conjectures, numerous random theories can be generated, and the conjecture discarded only if it is true of all of them. However, it is worth pointing out that the sparsity of data in the random counterpart theories is inherited from the original data. Hence, the original data suffers from the same likelihood of a conjecture being true purely because the construction process narrows down already sparse data. As an example from the animals domain, take the conjecture: $legs(a, 0) \wedge homeothermic(a) \leftrightarrow covering(a, none) \wedge milk(a)$, which states that all leg-free, warm-blooded animals have no covering and produce milk (and vice-versa). This may appear interesting, but it is likely to be very brittle as a general result, given that there is only one animal in the dataset (the dolphin) for which either the left-hand or right-hand concept applies. Hence, while this conjecture is not tautologous, throwing it away because data sparsity led the random theories approach to suggest it as a tautology, would probably not be disastrous. In [11], we say that such conjectures have *low applicabilities*, which can indicate low interestingness.

We have found the random theories approach to be highly effective at removing all tautologous implications, equivalences and non-existence conjectures, while removing relatively few non-tautologous ones. Hence, when coupled with other filters which look at the background concepts discussed in a conjecture, we reduced the volume of uninteresting conjectures shown to users very well. Moreover, even when hundreds of random theories are used, the process is remarkably fast because it is usually only applied to a small number of user-chosen conjectures. Note also that the approach is formalism independent, i.e., we do not need to resort to first order theorem proving or other techniques which would limit the type of concepts HR3 can produce. Moreover, as it uses only existing ATF techniques, this method is low maintenance and sustainable as the number of production rules in HR3 increases.

Tautologous conjectures are only part of the problem. Another major problem is with redundancy in the output conjectures. For instance, if we are presented with the conjecture $\forall\, t\,(A(t) \rightarrow B(t))$, then we do not want to be presented later with the conjecture $\forall\, t\,(A(t) \wedge C(t) \rightarrow B(t))$, as this latter one is entailed by the former. HR2 uses a slow and incomplete forward chaining method to remove redundancy, and we have also appealed to third party theorem provers when the conjectures were expressible in first order logic. Hence, neither of these approaches was taken forward into HR3, and we turned again to the idea of using randomly generated theories.

Given a pair of implication conjectures $C_1 : L_1(t) \rightarrow R_1(t)$ and $C_2 : L_2(t) \rightarrow R_2(t)$, as before, we generate a number of random theories as a test-bed for the hypothesis that C_1 entails C_2, that is, if C_1 is true, then C_2 must be true as well. As C_1 comes from an original theory, the procedure required to produce concept L_1 is known and can be applied to the random theory. After this, the datatable for the concept with definition $((L_1(t) \wedge R_1(t)) \vee \neg L_1(t))$ is constructed. This is done by HR3 using its own production rules, namely a *conjunction* step followed by a *negation* step and finally a *disjunction* step. The tuples in the resulting datatable are those which support conjecture C_1, i.e., they are tuples, t, for which $L_1(t) \rightarrow R_1(t)$ is true in the random theory. A similar construction of the datatable for $((L_2(t) \wedge R_2(t)) \vee \neg L_2(t))$ is performed, resulting in a second set of tuples. If the set of tuples for the first construction is a subset of those from the second, the most likely reason is that the two are related by the nature of the production rules. If we use multiple random theories as before, the likelihood of this explanation increases. Hence, we know that any tuple supporting conjecture C_1 also supports C_2, and we can infer that C_2 is entailed by C_1, so C_2 can be discarded without loss of information. We have implemented similar methods for non-existence and equivalence conjectures, and we use these to remove conjectures in the second case study below.

System	L3A1	L3A2	L3A3	L4A1	L4A2	L4A3	L5A1	L5A2	L5A3	L6A5	L6A6	L6A7	Av.
HR2	1920 (6)	3840 (22)	25960 (56)	1960 (10)	3970 (12)	25030 (37)	2080 (10)	4570 (15)	15090 (29)	29570 (45)	30250 (75)	40880 (75)	15427 (33)
HR3	40 (0)	60 (0)	90 (0)	50 (0)	70 (0)	120 (0)	80 (0)	120 (0)	170 (0)	550 (0)	570 (0)	2180 (0)	342 (0)
Progol	50 (15)	80 (17)	100 (17)	40 (13)	50 (13)	140 (14)	40 (16)	70 (16)	160 (16)	1430 (21)	3990 (25)	4920 (31)	923 (18)

Table 2. Execution times in milliseconds for HR2, HR3 and Progol running on a 3.4Ghz processor, averaged over 100 investigation problems requiring solutions of the form LXAY (with exactly X literals of arity at most Y). Percentage error rates are given in brackets.

5 Dynamic Investigation Problems

We use the term *investigation problem* for a type of problem which models to some extent a generic situation which might arise in, say, medical diagnosis or the solving of a crime. That is, there are a number of possible diagnoses/suspects, and the problem is to use the facts of the case to rank them in order of increasing likelihood of being the cause of the illness/guilty of the crime (which we call the *target* candidate). Such ranking often leads to further medical tests/police enquiries focusing on the most likely *candidates*, which will bring to light further information about the current case. Hence, we use the term *dynamic investigation problem* (DIP) to describe a series of such problems to be solved, as detailed in [26]. Solving each problem entails using the facts of the case, coupled with prior knowledge about the domain to narrow down the candidates to just one. Hence, a natural way to model such problems is as a constraint satisfaction problem (CSP), with one variable which takes one of n values, each representing a candidate, and the facts of the case acting as the constraints. Solving the case means finding a value to assign to the variable which doesn't break the constraints.

Often, however, not all the essential information is readily available, hence these problems are best modeled as partial CSPs. As such, especially during the early stages of the investigation, there will be no outright solution, and the constraints in the CSP need to be used to rank the candidates for further investigation. Additional relevant information can often be found in related past cases, from which regularities can be observed and utilised, and consultation of previous case studies is part of the investigation process. To model this, each investigation problem is given as a pair (C, B), where C is the CSP expressed as a Prolog program, and B is a Prolog background knowledge file, as would be found in an ILP application. Solving the investigation problem involves extracting rules from B which can be interpreted as extra constraints in C that enable a ranking of the candidates such that one is ranked higher than all the others.

For our study here, we have focused on the extraction of additional constraints during the middle of an investigation, i.e., when the constraints may not be enough to narrow down the candidates to the target directly, hence they should be used to rank the candidates in terms of their likelihood of being the target. One obvious way to do this ranking is in terms of how many constraints are upheld by each candidate. Given this, we see that a larger number of constraints would make the ranking more fine-grained than a smaller number, and this would increase the likelihood of one or two candidates being ranked higher than others. For instance, if there were 10 candidates and only two constraints found for the case, there would only be three classes of candidate: those satisfying 0, 1 or 2 constraints, and we would expect the number of candidates satisfying both constraints (hence being the most likely at this stage to be the target), to be 3 or 4. This is because we would expect the 10 candidates to be partitioned into 3.3 sets of 3. If, however, there were 5 constraints found, then we would expect the number of most-likely candidates to be 1 or 2, which would help narrow down further investigations.

We have developed a problem generator that can produce pairs (C, B) where B embeds a set of constraints, E, as a single clause, with a given number of literals, L, conjoined in the clause, and a given maximum arity, A, for the literals. The embedding is such that the clause is true of each of the target candidates in a number of case studies, and hence a 100% rule of the form:

$$target(X) \leftarrow A_1(\_, \dots, X, \dots, \_) \wedge \dots \wedge A_L(\_, \dots, X, \dots, \_)$$

can be mined from B. Each A_i here is to be interpreted as an additional constraint on the target candidate X. Note that embedded in B may be other smaller conjunctions of constraints (literals) which are also true of all the targets in the case studies, e.g., $target(X) \leftarrow A_1(X, \_) \wedge A_2(\_, X)$ may be true in all the cases, even though E contains five literals. Given the value of higher numbers of constraints, the problem here is to mine the clause which is true of all targets *and* contains as many literals as possible.

We produced 100 problems with 12 different pairings of L and A, as per the header in table 2, and we have used this synthetic dataset to compare Progol, HR2 and HR3 when mining the target clause from the background data. For Progol, this is a straight-forward predictive learning task, where a logic program implying the target literal is to be learned. For HR2, we formed a theory by exhaustively using only the *exists* and *compose* production rules, and extracted equivalence conjectures where the left hand side was the target concept. From those, we chose the one involving the most literals. For HR3, we exhaustively applied the *existential* and *conjunct* production rules to the data, and similarly extracted equivalence conjectures. For each concept with data equivalent to the target, the set of definitions for it were generated via expansion to completion. From the entire set of definitions, we presented as the solution the one utilising the greatest number of background concepts (HR3 has no access to information about literals in logical definitions, etc.)

The results from the 1,200 problems are given in table 2. We see that both Prolog and HR are very fast, taking less than a second per problem on average. HR3's efficiency was improved using a search reduction technique that avoided conjoining concepts where both have fewer examples than the target. It also used a random theories approach to remove any tautologous equivalences, which slightly improved the efficiency in generating the definitions. HR3 performed the same amount of search (in terms of theory formation steps) as HR2 did, but was 45 times faster over all the problems, which rises to 288 times faster for problem set L3A3. The error rate for HR3 is zero, because it was able to expand the definitions of equivalent concepts to completion. In many cases, the unexpanded definitions involved fewer than the required number of background concepts, but because HR3 could use other equivalence conjectures to expand the definitions of the background concepts, the most specific solution could be found in each case. HR2 suffered from not having this ability, resulting in an error rate of 33%. Progol uses an Occam's Razor principle based on information content to choose the most general hypothesis, which meant that in 18% of the problems, it returned a solution involving fewer background predicates than required.

Setup				Conjectures formed and removed						Execution times (ms)		
System	ED	NRT	Steps	Conjs1	Conjs2	Conjs3	H. Remove	E. Remove	Required	ATF	Removal	Total
HR2	-	-	3778	21784	815	363	20969	462	9/9	15990	648205	664255
HR3	1	1	3689	1165	122	74	1043	48	5/9	285	2198	2483
HR3	1	10	3689	1165	119	76	1046	43	5/9	276	3057	3333
HR3	1	100	3689	1165	118	76	1047	42	5/9	280	11429	11709
HR3	2	1	3689	59031	2352	783	56679	1569	9/9	272	38508	38780
HR3	2	10	3689	59031	2145	851	56886	1294	9/9	301	58251	58552
HR3	2	100	3689	59031	2094	913	56937	1181	9/9	273	113275	113548
HR3	2	250	3689	59031	2092	928	56939	1164	9/9	289	184629	184918

Table 3. Mondex results, comparing execution times on a 2.6Ghz processor, and conjectures found by HR2 with those found by different setups of HR3.

6 Invariant Discovery in Formal Methods

Formal methods are mathematically rigorous techniques used in developing and verifying software intensive systems. They allow the construction of models using mathematical notation, so developers can reason about their correctness, but the development of high quality formal models is time consuming and difficult [29]. *Refinement* techniques have been developed to handle the complexity of modelling large systems: starting from an abstract representation, details are added incrementally towards a more concrete representation which is closer to implementation. Invariants play an important role in formal modelling, as they express properties of the system which must always be true, hence specifying invariants in a model ensures that certain properties are never violated. Invariants can be used during refinement to prove consistency over the behaviours of a concrete step and the abstract step it refines. Moreover, invariants prevent the introduction of errors when changes are made to a model; conversely, their absence increases the possibility of errors being introduced into a model when the system evolves. Automated invariant discovery is an area of much interest in formal methods, and many approaches have been explored, e.g., the Daikon system [14] uses invariant templates to analyse program execution traces, and Bolton uses the Alloy analyser to discover invariants for Z specifications [2]. High-level patterns of proof have also been effectively used in constraining the discovery of program invariants [16, 20].

We have used the HR2 system to automatically discover invariants of Event-B models in a refinement step of the Mondex model developed in [3], as described in [18, 19]. Mondex is a system for the transfer of money between electronic purses, where each transfer must follow a protocol ensuring that no money is lost in a transaction, regardless of its success or failure. For a particular refinement step, the developers in [3] underwent a manual iterative process of invariant strengthening in the search for the correct invariants to overcome refinement proof-failures. We simulated this process using three sessions with HR2 to extract invariants in the form of 100% true conjectures. The background concepts for HR2 were derived directly from the state of the model (i.e., sets, constants and variables), and the examples for each of these concepts were derived from *simulation traces*. Such simulation allows the analysis of the operation of a model by observing how its state changes when different scenarios are explored. As a result, a simulation trace represents a record of the behaviour of the system at each step of the simulation.

The production rules to use in HR2 were selected with a *proof-failure analysis* of the model, and HR2 was run for 1,000 steps to produce a theory from which various equivalence, non-existence and implication conjectures were extracted. Details of the proof failure analysis and how the conjectures were selected are given in [19]. The equivalence and implication invariant candidates were pruned using a heuristic method to remove any where a literal on the left hand side also appeared on the right hand side (or vice versa). The non-existence conjectures were pruned by removing any with a repeated literal. Following this, the Prover9 automated theorem prover [23] was used to remove any conjecture that was entailed by another. With each resulting conjecture, a formal proof was sought that it expressed something which was invariant at all stages of processing in the model. Additional manual heuristics were used to select the final set of 9 conjectures that represent the invariants.

For our study here, which, unlike in section 5, involves data from a real-world application, we undertook a version of the experiments in [19] with both HR2 and HR3, with various considerations to make a comparison as fair as possible. For HR2, we used the *disjunct* and *compose* production rules, and we used HR3's counterparts. Rather than 1,000 steps, we enabled HR2 to run to a complexity limit producing all concepts with up to four background concepts combined. This matched the usage of *disjunction* once in HR3, followed by *conjunction* once. Also, HR2 and HR3 were both setup to remove conjectures using the same heuristics, and the stage where Prover9 was used in the HR2 sessions was replaced by HR3 employing random theories to remove entailed conjectures. HR3 found all the required invariants in a single session, whereas HR2 was used in three sessions, and we averaged the statistics for these. We experimented with different setups for HR3, namely by altering the *Expansion depth* (ED) for definitions in conjectures, and the *Number of Random Theories* (NRT) used for pruning entailed conjectures.

The results from the experiments are given in table 3. We first note that the theory formation session times (ATF) for HR3 are around 57 times faster than for HR2. In addition, the times taken by HR3 to remove entailed conjectures (Removal) are much smaller than the time spent in the HR2 sessions using Prover9 for the same job. We note that, if speed is important, the ED=NRT=1 setup for HR3 finds 5 out of 9 of the (Required) invariants in around 2.5 seconds. Here, however, while it didn't remove any required invariants, the random theories approach did remove some conjectures which were not tautologously entailed by others. This number reduces when NRT is increased: we see in the (E. Remove) column that only 42 entailed conjectures, as opposed to 48, are removed when 100 random theories are employed. When definitions are expanded to depth 2, nearly 60,000 raw conjectures are formed (Conjs1), but these are rapidly reduced by the heuristic (H. Remove) to around 2,000 (Conjs2), and the random theories approach reduces this by around half again, to less than 1,000 (Conjs3). The increase in number of conjectures found by HR3 over HR2 is due to HR3 unpacking definitions more. As in the ED=1 cases, as the number of random theories is increased, the number of entailed conjectures removed reduces. It's clear that, even with 250 random theories, a number of non-entailed conjectures are being removed. However, it's worth re-iterating that the majority of these will be due to sparsity of data, and likely not particularly interesting. None of the required invariants were removed, even when only one random theory was employed (taking 38.8 seconds in total).

7 Conclusions and Future Work

This is the first report about the HR3 software, the latest implementation taking forward our investigation of Automated Theory Formation (ATF) as an approach for discovery tasks. ATF is an artefact generation method where entire theories are formed about a domain and then investigated for various purposes, rather than a problem solving approach where a single answer is sought. We provided design considerations based on successes and failures of HR2, and gave some details about how HR3 generates, stores and presents theories. Our two main design ideals were to: (a) allow HR3 to produce a variety of artefact types within a theory, and further variety within each type, and (b) build HR3 to be independent of any logical formalism. To this end, we described a new approach to the removal of tautologies and redundant material from theories, which uses randomly generated data against which to test the truth of conjectures.

We showed the value of HR3 and the new approaches in two case studies where we compared it against HR2. The DIP and Mondex examples were driving forces for the development of HR3. In particular, HR2's high error rate and slow speed in solving DIPs meant that we needed to employ other systems such as Progol and Weka for this task, and HR2's slowness for the Mondex application meant that any real-time usage of ATF for formal model designers was unlikely. We have omitted details of memory consumption in the case studies, but this was also a factor in the poor performance of HR2 for these tasks. In both case studies, we showed that HR3 is much faster at building theories with more value than HR2.

We continue to develop HR3 by adding production rules and functionality according to the design considerations presented here. The random theories approach has much potential, as it is formalism-independent, low maintenance and very effective at tidying up theories before presentation to the user. Increasing the number of random theories tested increases the accuracy of this approach in not removing false positives, and we have argued that many of these false positives, while not tautologous or redundant, are still likely to be uninteresting as they arise from scarcity of data, rather than the semantics of the data about which a theory is being formed. We plan to investigate the conjectures which are discarded thoroughly and to get HR3 to calculate and use a probability that a given conjecture is tautologous. We will also investigate whether increasing the size of random theories is more effective than increasing their number, in terms of accuracy and efficiency. Initial testing indicates that it will be faster, but there may be issues in multiplying the size of a random theory while maintaining the structure of the original.

We plan further case studies with HR3 to test its value as a discovery system. We have so far repeated HR2 experiments with HR3 on classic machine learning dataset such as mutagenesis [6] and on new problems, e.g., large databases of shopping information, ontologies about wine, etc. In addition, we have shown how the numerical productions rules in HR3 can be used in a distributed way to solve (in the sense that Checkers has been solved [27]) arithmetical puzzles such as Countdown [10], and we plan to generate new puzzles of this type. Finally, in unpublished work, we have successfully applied HR3 to the automatic generation of stanza structures for poems where the lines of each stanza are supplied from Twitter. Such applications demonstrate the value of HR3 as a discovery and creativity tool, and we hope that it will have more impact outside of our own work than HR2 did, due to the improvements described above. In particular, we expect to have an online version of HR3 running soon, with which people will be able to submit background information in a variety of forms and inspect the theories that HR3 produces.

Acknowledgements

This work has been supported by EPSRC grants EP/J004049 and EP/F036647, and through EC funding for the projects WHIM (611560) and COINVENT (611553) supported by FP7, the ICT theme, and FET program. We would like to thank the many people with whom we discussed the random theories approach, and the anonymous reviewers for their useful comments.

REFERENCES

[1] B Baars, *A Cognitive Theory of Consciousness*, CUP, 1988.
[2] C. Bolton, 'Using the Alloy analyzer to verify data refinement in Z', *ENTCS*, **137**(2), 2005.
[3] M. Butler and D. Yadav, 'An incremental development of the Mondex system in Event-B', *Formal Aspects of Computing*, **20**(1), 2008.
[4] J Charnley, S Colton, and I Miguel, 'Automatic generation of implied constraints', in *Proceedings of the 17th ECAI*, 2006.
[5] S Colton, 'An application-based comparison of Automated Theory Formation and Inductive Logic Programming', *ETAI*, **4(B)**, 2000.
[6] S Colton, 'Automated Theory Formation applied to Mutagenesis Data', in *Proc. of the 1st Anglo-Cuban Symposium on Bioinformatics*, 2002.
[7] S Colton, *Automated Theory Formation in Pure Math.*, Springer, 2002.
[8] S Colton, A Bundy, and T Walsh, 'HR: Automatic concept formation in pure mathematics', in *Proceedings of the 16th IJCAI*, 1999.
[9] S Colton, A Bundy, and T Walsh, 'Automatic invention of integer sequences', in *Proceedings of the 17th AAAI*, 2000.
[10] S Colton, 'Countdown: Solved, Analysed, Extended', in *Proceedings of the AISB Symposium on AI and Games*, 2014.
[11] S Colton, A Bundy, and T Walsh, 'On the notion of interestingness in automated mathematical discovery', *IJHCS*, **53(3)**, 2000.
[12] S Colton and S Muggleton, 'Mathematical applications of Inductive Logic Programming', *Machine Learning*, **64**, 2006.
[13] S Colton and G Wiggins, 'Computational Creativity: The final frontier?', in *Proceedings of the 20th ECAI*, 2012.
[14] M Ernst, J Perkins, P Guo, S McCamant, C Pacheco, M Tschantz, and C Xiao, 'The Daikon system for dynamic detection of likely invariants', *Science of Computer Programming*, **69**(1–3), 2007.
[15] M Hall, E Frank, G Holmes, B Pfahringer, P Reutemann, and I Witten, 'The WEKA data mining software: An update', *SIGKDD Explorations*, **11(1)**, 2009.
[16] A Ireland, B Ellis, A Cook, R Chapman, and J Barnes, 'An integrated approach to high integrity software verification', *Journal of Automated Reasoning*, **36**(4), 2006.
[17] I Lakatos, *Proofs and Refutations*, CUP, 1976.
[18] M T Llano, *Invariant Discovery and Refinement Plans for Formal Modelling in Event-B*, Ph.D. dissertation, Heriot-Watt University, School of Mathematical and Computer Sciences, 2013.
[19] M T Llano, A Ireland and A Pease, 'Discovery of invariants through Automated Theory Formation', *Formal Aspects of Computing*, 2012.
[20] E Maclean, A Ireland, and G Grov, 'The CORE system: Animation and functional correctness of pointer programs', in *Proceedings of the 16th IEEE Conference on Automated Software Engineering*, 2011.
[21] W McCune, 'The OTTER user's guide', Technical Report ANL/90/9, Argonne National Laboratories (ANL), 1990.
[22] W McCune, 'A Davis-Putnam program and its application to finite first-order model search.', Tech. Report ANL/MCS-TM-194, ANL, 1994.
[23] W McCune, 'Prover9 Manual' available at `http://www.cs.unm.edu/~mccune/prover9`, 2009.
[24] S Muggleton, 'Inverse entailment and Progol', *New Generation Computing*, **13**(3–4), 1995.
[25] A Pease, S Colton, and J Charnley, 'Automated theory formation: The next generation', *IFCOLOG Journal Proceedings in Computational Logic, Special Issue on Theory Exploration*, (to appear) 2014.
[26] R Ramezani and S Colton, 'Automatic generation of dynamic investigation problems', in *Proc. of the Automated Reasoning Workshop*, 2010.
[27] J Schaeffer, N Burch, Y Björnsson, A Kishimoto, M Müller, R Lake, P Lu, and S Sutphen, 'Checkers is solved', Science, 317(5844), 2007.
[28] V Sorge, A Meier, R McCasland, and S Colton, 'Automatic construction and verification of isotopy invariants', *Journal of Automated Reasoning*, **40(2-3)**, 2008.
[29] J Woodcock, P G Larsen, J Bicarregui, and J Fitzgerald, 'Formal methods: Practice and experience', *ACM Computing Surveys*, **41**(4), 2009.