Creative Logic Programming

Simon Colton
sgc@doc.ic.ac.uk
Imperial College, London
180 Queens Gate, London, SW7 2BZ. United Kingdom

Abstract

The standard machine learning paradigm is to find something that users know they are looking for, with the discovered artefact defined in terms of given background knowledge. We propose to extend this to automating the task of finding novel and interesting information – also based on the background knowledge – that the users *do not* know they are looking for. We sketch various methods for introducing knowledge to a knowledge base which are inspired by notions from the study of creativity. We attempt to determine situations where it is possible to project certain words from the creativity literature onto an agent (human, machine or otherwise), as it undertakes the task of adding information to a knowledge base. This study has enabled us to suggest a road-map for the development of creative logic programming systems, which extends inductive logic programming approaches to discovery tasks.

1 Introduction

An accepted view of machine learning is given in [24]:

> An [agent] is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Usually, the tasks in T are prediction tasks. That is, machine learning programs are given some background information, and a categorisation of a set of examples. They are then required to determine a method for predicting into which category an unseen example should be placed. The learned method will normally involve aspects of the background information (experience), and may be any number of things: a classifying concept, a set of rules, or a mathematical function such as a neural network. In this general setting, if we think of the task as learning a target function, then it is clear that, in some respects, while the exact details of the target are not known, at least something is known about it: the way it categorises the given examples. Hence, in many respects, the user knows what they are looking for, but they do not know what it looks like.

As an example, suppose the Progol [26] machine learning program was given a single background concept: of one integer dividing another. Suppose also that it is given positive examples of the target concept as 1, 4 and 9 and negative examples of the target concept as 2, 3, 5, 6, 7 and 8. It would very easily learn the concept of square numbers to explain why 1, 4 and 9 are positive examples and the other numbers are negative. This is an illustrative example where the user knew what they were looking for, and so it was possible to use a goal-based approach such as Inductive Logic Programming (ILP) which underpins Progol, as discussed in [25]. ILP systems specifically designed for this task are said to be performing *predictive ILP*.

We are interested here in a more general task than prediction, namely, to start with a given background knowledge base and use this as a seed from which to build a larger knowledge base. This will be done in such a way that the learned knowledge base will contain a high proportion of information which is both *novel* and *interesting* to the user. In this way, we want to extend the task of finding things the user knew they were looking for into the task of finding novel and interesting things (facts, concepts, artefacts, etc.) that the user did not know they were looking for. ILP systems such as CLAUDIEN [14] and WARMR [15] perform what is known as *descriptive ILP*. These aim to discover association rules, which relate predicates from the background knowledge. Hence, these are more likely to find something the user didn't know they were looking for than predictive ILP systems. However, the user often specifies a language bias which effectively restricts the format of the rules which can be found. So, the user may not know what, if any, task the rules found by the system will solve, but they will know the format the rules will take. Hence, these systems still have some way to go before being able to truly approach the kinds of discovery tasks we are proposing.

The HR system [9] also undertakes descriptive ILP, but its output is a theory, which will contain concepts (classification rules in ILP terminology), conjectures (association rules), constants and proofs which were not present in the background knowledge. HR has mostly been used for applications in mathematical domains, but we have recently applied it to other scientific datasets, in particular the mutagenesis data [8]. HR differs in many ways with other predictive ILP systems, most notably because it interacts with third party

systems, including automated theorem provers such as Otter [20], model generators such as MACE [21] and computer algebra systems such as Maple [1]. HR has been used for prediction tasks, such as those described in [10] and [22]. However, it is has mainly been designed to perform exploration of a domain in order to build up a theory, i.e., to find facts, examples, hypotheses, etc., about the background knowledge that the user did not necessarily know they were looking for.

For instance, given only the divisors concept as background information, HR invents concepts such as refactorable numbers, which are such that the number of divisors is itself a divisor. Moreover, HR finds interesting conjectures about these new concepts: for example, HR makes the conjecture that odd refactorable numbers are always square numbers, a fact we prove in [3]. As another example, in experiments described in [9], we gave HR just the axioms of various algebraic domains, which usually amounted to just two or three lines in a background knowledge file. HR then used MACE to generate models for the axioms, and used Otter to prove theorems about them, which led to HR forming rich theories about the algebraic structures. In a later application, a similar approach was used to discover implied constraints in order to reformulate constraint satisfaction specifications for quasigroup existence problems [11].

Of course, exploration such as that undertaken by HR may lead to very dull information being discovered. Hence the aim of automated theory formation, as expounded in [9], is to find novel, interesting facts about the domain concepts supplied by the user. This is exactly how we have phrased the extension of machine learning tasks above. However, we make no claim that the HR system covers every possible way in which an agent could build a knowledge base from a seed of background information. Indeed, the work presented here shows that the techniques used by HR are only a small part of a much larger set of techniques which it could potentially employ. However, we do take HR as a role model in two respects. Firstly, it is able to discover different types of artefact: as we've mentioned, the interesting output from a theory formation session might be a concept such as refactorable numbers, or a rule (conjecture) such as odd refactorable numbers being square or an example such as 2, which is the only prime refactorable number. Secondly, HR combines various forms of reasoning. It uses invention and induction itself to produce concept definitions and make hypotheses about them. It also uses deduction and calculation when it appeals to Otter and MACE to prove/disprove a hypothesis that it has made. With an extension described in [12], HR is also able to use abduction to fix faulty hypotheses.

Note that standard prediction tasks fit into our extended framework. In this case, while the agent may come into contact with a large number of concepts and hypotheses as it searches, the output is usually only a single way of completing the precise task set down, such as a single neural network, or a small set of rules governing the prediction task. The interestingness of the learned knowledge is tested with respect to the original task, and the new knowledge may be uninteresting if it has a low predictive accuracy when assessed by a method such as 10-fold cross validation. Also, knowledge learned for prediction tasks may be uninteresting if it can be shown to be overfitting (memorising) the data, as described in [24]. Note that *boosting* machine learning approaches aim to combine many learned prediction methods in order to better achieve prediction of unseen examples [28]. However, the combined method is still assessed in terms of overfitting and accuracy with respect to the single prediction task.

Our extended task specification covers other intelligent tasks which may be of interest to a user who has some data. Such tasks include the generation of puzzles [7], which we argue in [4] is difficult for a goal-based machine learning approach. Such tasks also include "proactive" rather than "reactive" machine learning. Proactive machine learning involves learning possible answers to machine learning questions before the details of the actual question are given. For instance, in [5], we propose the following task, based on Michalski's train problem [23]: given a set of 20 trains – described using concepts such as carriages, contents of carriages, wheels, etc., – and given, say, an hour, prepare to answer with no hesitation a classification task. The classification task will involve being told that a particular set of 10 trains are going East and 10 trains are going West, and the question is to determine why the trains are going in their respective directions. As the 10 Eastbound trains are not known beforehand, the learning agent has to invent and solve possible classification tasks in the hour it is given, in preparation for the actual test supplied later. This is therefore a fundamentally different problem to the original trains problem, which is a standard classification/prediction task.

In the remainder of the paper, we will look at various possible methods by which an agent (human, machine or otherwise) could add information to a knowledge base in order to possibly identify something the user did not know they were looking for. Our inspiration comes from notions of creativity: we will look at a word or phrase (such as deduction, induction, serendipity) and attempt to give at least one situation where it would be possible to project the word/phrase onto the actions taken by an agent to add new information to the growing knowledge base. We hope to do so in such a way that it is difficult to argue that the notion does not apply to an agent performing in the way prescribed. For example, if we look at the word "deduction", often mentioned in creativity studies, it is uncontroversial to say that an automated theorem prover using the Modus Ponens rule of inference to generate new knowledge has reasoned deductively.

Note that we **do not** intend to say that *only if* an agent performs in a certain way can a particular notion associated with creative behaviour be projected onto it. Rather, we want to be certain that we have identified a situation in which the notion applies. It is for this reason that we have chosen to undertake this study via a cut-down representation language such as logic programs. By giving particular simple instances where the amount and type of information is clear, we hope to approach the study of creativity from the bottom up. This is in contrast with other approaches to the study of creativity, which have approached from the top-down, by characterising various thought processes, especially in ad-hoc case studies, such as Kekule's discovery of the benzene-ring structure [2]. We hope that our bottom-up approach will compliment these top-down approaches and go some way to de-mystifying vari-

ous notions from the study of creativity, bringing a more concrete understanding of this topic.

In section §2, we present some techniques for introducing new knowledge to a knowledge base. The techniques are presented as situations where an agent takes existing knowledge, represented as logic programs, and generates new knowledge. Moreover, the techniques are inspired from notions arising from the study of creativity, and the situations are designed so that it is possible to project certain words from the creativity literature (induction, serendipity, etc.) onto the agent as it carries out the technique. Note that it is not our intention to provide acute details of how new knowledge may be introduced. This is planned for the next stage of our study into creative logic programming, where we analyse various techniques such as ILP, resolution proving, constraint satisfaction, etc. in order to interpret them within the same framework. Given that we intend this framework to use logic programs as its underlying representation, it is important that we use logic programs in our study here, so that we maintain consistency.

In §3, we supply a worked example – the mutilated checkerboard problem – and give four scenarios in which a program might use the techniques presented in §2 to solve this problem automatically. We use the scenarios to determine the extent to which we can project the word creativity onto an agent. We conclude by presenting a road-map for future work in creative logic programming, based on the framework presented and using the Mutilated Checkerboard as an inspiring example [13].

2 Generating New Information

The purpose of this first report about creative logic programming is to give an overview of many methods which could be applied, rather than to go into the technical details of a few techniques, some of which is covered elsewhere (for instance in the various descriptions of inductive logic programming). In later reports, we intend to provide more specific details of these methods, using the formality afforded by logic programs.

At present, it suffices to say that we are proposing a situation whereby a user has some background knowledge expressed as logic programs, as they would in an inductive logic programming application. The task of the agent is to introduce new knowledge into the knowledge base in the hope that something introduced will be novel and interesting to the user, and embody some piece of useful information they did not know they were looking for. We do not look in detail here at the notion of interestingness. However, it should be noted that the methods detailed below are inspired by notions from the creativity literature. Hence it is likely that these methods (or something similar to them) have been identified as techniques which have, at some stage, been used for the introduction of something interesting as part of a creative process.

We present below distinct situations in which an agent could add knowledge to a knowledge base. It is important to note two points: (i) these situations are not meant to constitute an exhaustive list of all the ways in which a creative agent could add information to a knowledge base and (ii) we are in no way attempting to characterise the creative notions given below, rather we simply want to present situations in which it is possible to argue that the word applies to the agent's actions. At present, the situations are sketches, but in future, we hope to appeal more to the formal aspects of logic programming in order to make the framework more concrete.

Logic programs are a subset of first order predicate logic, where a logic program is a set of Horn clauses. A Horn clause can be thought of as an implication conjecture where a conjunction of *body* literals imply a single *goal* literal. A literal can either be a predicate, a function or a constant. In the discussion below, we often use the word 'hypothesis' for a Horn clause, as this is used in many logic programming contexts. Also, we use the word 'concept' for a clause of the form:

$$concept_C(X_1, \ldots, X_n) \leftarrow p_1(\ldots) \wedge \ldots \wedge p_m(\ldots)$$

These define a property of tuples of constants. We say that one logic program *entails* another if the latter is a logical consequence of the former.

- Experimentation

Suppose an agent has a Horn clause H and it doesn't know whether H is true or false. However, it does know the types of the variables in H, and knows how to generate examples of the types, so can generate test tuples for the Horn clause. Note that this generation process might be as simple as looking them up in the knowledge base. Suppose further that the agent tests H against these test tuples in order to find a tuple for which the body of the clause is true, but the head is false. Such a tuple would show that H is false, and we say that the agent is performing experimentation.

- Deduction

Suppose an agent has a Horn clause H which it knows to be true, either because it is an axiom, or it has proved it (see below). In this case, there are various logic program which are *entailed* directly from H. These can be obtained by passing H – possibly along with other Horn clauses known to be true – through one of many rules of deduction, such as Modus Ponens, Modus Tollens, Resolution, etc. The resulting Horn clause is also known to be true because deductive rules of inference only produce true clauses (if they are supplied with true clauses). We can define a deduction step as comprising a set of Horn clauses known (or at least temporarily assumed) to be true, a rule of inference and a Horn clause derived by applying the rule to the true clauses. Such steps comprise the operators in the search carried out by automated theorem provers, and an agent using them can certainly be thought of as performing deduction.

- Proving

Suppose an agent has a Horn clause H and it doesn't know whether H is true or false. Suppose further that it finds a series of deduction steps where the final one outputs H from a rule of inference. Note that the inputs to the inference rule may have also been the output from a previous step, but that each step was based on Horn clauses known to be true (either

because they are axioms, or they were deduced from something known to be true). The series of deduction steps comprises a proof of H, and the output from each can be thought of as adding new knowledge to the knowledge base. Note that there are alternative ways to prove H, most notably by negating H and using the resolution method to derive a False clause.

In another situation, suppose the agent knows that the set of examples to which H applies is finite, and it exhaustively checks whether the hypothesis is true of every example in the list. If the hypothesis is true of every example, then the hypothesis cannot possibly be false, hence it must be true. We call this technique proving by *exhaustive experimentation.* Any agent which finds proofs of unknown hypotheses can be thought of as performing the act of proving.

- Induction

Suppose an agent invents a new concept and then proceeds to check each known concept in the knowledge base to see whether it has exactly the same success set as the new concept. Note that the success set of a concept is the set of tuples of ground instances which satisfy the definition of the concept. If the agent finds such a match, then it could make the hypothesis that the equality is true in the general case, rather than just a coincidence due to the limitations of the knowledge base it was supplied with. An equivalence conjecture between two concepts could be represented as a set of Horn clauses, by looking at the two implication conjectures embedded in the equivalence and extracting Horn clauses by taking the left hand side to imply the first literal of the right hand side, then the second literal, and so on.

This kind of inductive reasoning is similar to that carried out by the HR system. HR can also make implication hypotheses if it notices that the examples of one concept are all examples of another, and non-existence hypotheses when it finds that an invented concept has no examples. Similar kinds of inductive reasoning occur in the Progol ILP system. In this case, a set of positive and negative examples are generated and rules of inductive inference are used to generalise hypotheses. Each generalised hypothesis is checked to see if it explains why the positives are positive and the negatives are negative. If the hypothesis is a perfect match, then the system induces the rule that this is true in the general case, and returns this answer to the user.

- Abduction

There are various interpretations of abduction. We shall think of it as a special case of induction, whereby a possible explanation for a phenomenon is not so much generated as found within the information it has available already. Hence, suppose we have a situation where an agent has found a hypothesis, H, for which it does not know the truth. The agent then looks through a set of hypotheses it already has (not necessarily just the true ones), and finds a hypothesis, G, which entails H. In this case, it has abduced G as a possible explanation of H. There are abductive logic programming systems [16], and we are currently extending HR to use abduction [12].

- Exploration

Suppose, rather than a set of examples, an agent was given a logic program able to generate examples. One way in which we could possibly project the word exploration onto the agent was if it used this generation method to produce a set of examples in order to provide empirical evidence for inducing various hypotheses. Note that there are many more ways in which an agent could perform exploration, for example the successive use of invention (see below) can be deemed exploration of a search space. Also, exploratory creativity is characterised by Boden [2].

- Invention

The phrase 'necessity is the mother of invention' is certainly true in many cases. Suppose an agent takes two Horn clauses G and H, and passes them through the intra or inter-construction rule of inductive inference as described as described in [26]. These rules perform predicate invention, which introduces a new predicate symbol. In this case, the purpose of the invention is to determine a Horn clause from which G and H would follow using a resolution inference.

Possibly curiosity is not as closely related to invention as necessity is, but it is certainly possible to invent things for no apparent reason at the time of invention, then explore any implications or applications of the invention. Such invention is performed by genetic algorithms, and we can use this observation to propose the scenario where an agent takes two Horn clauses, G and H, and randomly chooses literals from the body of G and literals from the body of H to form the body of a new Horn clause, with the head of the new clause chosen randomly also. Such crossover techniques are integral to genetic algorithms, and there are now some genetic logic programming implementations [17]. Note that genetic techniques also include mutation, which is another inventive technique.

Invention for curiosity alone can be done in a more principled way than randomly producing new Horn clauses. The HR system implements such a principled approach through a set of 12 production rules, which take old concepts and produce new ones. For instance, thinking of the old concepts as Horn clauses, HR's compose production rule produces the Horn clause obtained by taking all the literals from two old Horn clauses, discarding any repetitions. Note that the compose rule can also perform unification steps while combining the Horn clauses. Such invention steps are performed purely in order to explore the domain, and not for any particular problem-solving reason (except the overriding one – to find something interesting in the domain). For a more detailed discussion of the production rules, see [10].

- Innovation

The word innovation has some loose notion of quality associated with it: many things can be invented/discovered, but if they are not of value, or do not improve upon the state of the art, it is difficult to project the word innovative onto the process which produced them. We can see that such innovation

occurs in genetic algorithm approaches, where the measure of value is the fitness function. Similarly, HR has many measures of interestingness which it uses to assess the value of inventions, in order to know when an innovation has occurred and to exploit that innovation (by building new concepts using it – a heuristic which is intended to increase the value of the theory as a whole).

Hence the following situation is one in which an innovation occurs: suppose an agent has a measure of value, M, for a concept, and invents a new concept D, where D is based on old concepts B and C. If D is measured to be of higher value with respect to M than both B and C, then D is an innovation. We can also supply a more concrete situation: suppose an agent has a Horn clause, H, which it is trying to prove. H is not entailed by the current knowledge base, but the agent invents a new concept and induces a hypothesis, T, about the new concept, which it proves. T now makes it possible to prove H. Clearly, T is an innovation, because the ability of the agent to prove H is increased by the discovery of T.

- Exploitation

We want to capture here the notion of making use of a chance occurrence. Such situations occur in human life when people have many tasks to perform (or problems to solve) and, while in the pursuit of completing one task, something arises which allows them to complete another task. If we say that in the task of adding to the knowledge base, one problem faced by an agent is to determine the truth of Horn clauses, then we can suggest the following situation: the agent has a set, S, of Horn clauses for which it does not know the truth. While experimenting in order to disprove a particular hypothesis, H, from S, it generates an example which may or may not disprove H. However, it checks whether the example falsifies the other Horn clauses in S, and finds one for which it does. We say that the agent has exploited the introduction of a new example in this situation. Note that the HR program performs this kind of exploitation: whenever it generates a counterexample to a false conjecture, it checks to see whether the same counterexample breaks any other open conjectures. A similar situation may occur if the agent first deduces something new, then checks whether the knowledge base including this new addition entails one of the unknown Horn clauses in S.

- Serendipity

If a situation is said to be serendipitous, this is often misinterpreted to simply mean lucky, which is a simplification of the true meaning of the word. In particular, serendipity involves not just exploiting a new piece of knowledge, but rather creating a situation in which a new piece of knowledge may be exploited. This knowledge may be found by chance (invention, etc.) or it may come from a reasoning process (deduction, etc.). Hence we believe that in the following situation, it is possible to say that an agent has acted serendipitously. Suppose an agent has recently found a Horn clause, H, which it knows to be true. Suppose further that there is a set, A, of Horn clauses known to be true, which contains H. It then constructs a hypothesis X, which is entailed by A, but which is *not* entailed by A/{H}, i.e., set A with H removed. That is, X can only be proved true if H is known to be true. Here, given the (partial) solution H, it has constructed the problem of proving a hypothesis X, and has thus acted serendipitously.

- Imagination

Suppose an agent has invented a concept for which it can find no examples. It then introduces a new ground term, T, (or set of ground terms) and states as an axiom, A, that T satisfies the definition of the concept. It continues by using A in various deduction steps in order to determine which other concepts, if any, are satisfied by T. In this situation, we can say that the agent has imagined T and determined some additional properties of T. Note that this may generate a contradiction, leading to the proof that such a T does not exist. For instance, the field of complex numbers in mathematics is based on such an imaginative step, namely the imagination of a number which multiplies by itself to give -1. Appropriately, this number is symbolised by the letter i, and is called an imaginary number.

In another situation, suppose an agent has a hypothesis, H, for which it does not know the truth. If it then assumes the truth of H, and either deduces something from it directly, or uses H in the proof of another hypothesis, then we could say that it has imagined the consequences of H being true. A similar situation would occur if the agent assumed the falsity of a theorem/axiom it knew to be true, or vice-versa.

- Analogy

The treatment of analogy will probably be better served by a situation in which an agent is working in two domains, or multiple agents are working in different domains, etc. However, for consistency, we will use the situation where a single agent is aiming to develop a knowledge base. Suppose that a new Horn clause, G, has been produced. Then, the agent looks through the set of proved Horn clauses and chooses one, H, which looks most similar in some respect to G. It then proceeds to carry out the same set of deductive steps as those used to prove H, perhaps applying similarity tests where necessary to decide which clauses to use in the deductive steps. We would say here that the agent has used analogy with a known result to suggest a way to proceed in a new situation.

- Reparation

Suppose an agent found a Horn clause, H, for which it could find a small set of counterexamples. It then altered H – effectively producing a new hypothesis H' – in such a way that none of the counterexamples to H were counterexamples to H' (and none could be found elsewhere). In this sense, the agent has repaired H. Note that such techniques for fixing faulty conjectures are prescribed by Lakatos in [18]. Moreover, there is a project underway to add such abilities to a multi-agent version of the HR system [27].

3 The Mutilated Checkerboard Problem

In [19], John McCarthy presents the Mutilated Checkerboard Problem as a *Drosophila*[1] for studying creative solutions to problems. The problem is stated as follows:

> *Two diagonally opposite corner squares are removed from a checkerboard. Is it possible to cover the remaining squares with dominoes? A domino is a 1 × 2 rectangle that can cover two rectilinearly adjacent squares.*

This problem has a long history in AI, as discussed in [19]. On a checkerboard, diagonally opposite corners have the same colour, so on a mutilated checkerboard, the number of squares of one colour is 32, while the number of squares of the other colour is 30. Humans usually solve the problem with the realisation that a domino covers both a black and a white square. That means that, no matter how many dominoes are put on the board, the number of uncovered black squares will always be different to the number of uncovered white squares. Hence, it is not possible to cover the entire board, because this would lead to the situation where the number of uncovered black squares equals the number of uncovered white squares (both equalling zero).

In [19], McCarthy proposes an informal definition for creative solutions:

- **Definition (informal)** A solution to a problem is creative if it involves concepts not present in the statement of the problem and the general knowledge surrounding it.

He further asserts that the standard solution to the mutilated checkerboard problem is creative because it involves concepts not present in the formulation of the problem, namely the concept of black and white squares.

We suppose that a user is employing a creative logic programming agent to solve this problem. We supply four scenarios in which an agent attempts to solve the problem. This enables us to provide overviews of different ways for the creative techniques described above to be fruitfully employed. The actions of the creative agent in each scenario are described in terms of the framework presented in §2. In each scenario, we pose and attempt to answer two questions: has the problem been solved, and how creative has the agent been?

3.1 Scenario 1

In this scenario, an agent is given logic programs which represent rules for (a) (virtually) placing dominoes on a board (b) stopping when it cannot place any more on the board and (c) checking whether all the squares have been covered or not. It is also given a hypothesis that it is not possible to cover the board with dominoes. It then exhausts all possible ways of placing a set of dominoes on a mutilated board, and in each case it observes that some squares remain uncovered, and reports this to the user.

It has clearly performed experimentation with respect to the hypothesis that it is not possible to cover the board with dominoes, because it has tried to find a counterexample to the hypothesis. This has resulted in proving the hypothesis via exhaustive experimentation. There is no doubt that the program has solved the problem: it has demonstrated that no covering exists. However, it is difficult to project the word creative onto the agent's actions. Part of the reason for this is that the agent hasn't found something the user didn't know they were looking for. McCarthy would point out that in the solution, no new concept has been invented. Also, the only novel things produced are examples of boards with dominoes on, which is unlikely to be seen as quality output, and certainly not something that is interesting to the user. Another part of the reason may be the nature of the calculation, which was fairly routine, involving only three processes. The user would probably believe that, if they had enough time, they could easily solve the problem in the same way, and the agent has not told them anything interesting other than a verification that the hypothesis they made was indeed true.

3.2 Scenario 2

It is possible to argue that the problem is posed in such a way that it enables people to *abduce* the creative part of the solution. Because the word checkerboard is stated in the problem formulation, this obviously brings to mind actual checkerboards. Of course, most people solve this problem without ever resorting to physically putting dominoes on a checkerboard. However, they are aware that a checkerboard (as opposed to an 8×8 grid) is coloured with black and white squares. Hence, it is not implausible to believe that the concept of black and white squares is part of what McCarthy calls the general knowledge surrounding the problem, and that humans do not invent the concept of black and white squares.

In light of this observation, in this second scenario, a creative agent is supplied with a plethora of background information about checkerboards and dominoes. This information would include concepts (such as black and white squares) and axioms, such as the fact that a domino always covers both a black and a white square. The agent would also be supplied with information about possible solutions to the covering problem, including the fact that if a full covering existed, there would be no uncovered white squares and no uncovered black squares. Finally, the agent is given the hypothesis that the board *cannot* be covered by dominoes.

Suppose the agent used rules of deductive inference to reason from the axioms to the hypothesis that the board cannot be covered by dominoes, hence proving by deduction that it is not possible. Clearly, the problem has been solved by a classical automated reasoning approach to the problem. This approach is possibly similar to what McCarthy had in mind when he originally posed the problem (in a Stanford AI memo entitled "A Tough Nut for Theorem Provers").

One could argue that the agent in this scenario has been more creative than that in scenario 1, because (a) it has produced something novel and of value, namely the set of deduction steps comprising the proof and (b) it is probable, although not certain, that the amount of calculation performed by the agent in this scenario is less than that performed by the agent in scenario 1 in amount, but greater in complexity. That is, the amount of choice of both what to deduce some-

[1] So called after the Drosophila Melanogaster – the fruit fly – which has been one of the most studied organisms in genetics.

thing new from and of which rule of deduction to perform the inference with is often quite high in automated theorem provers. Hence, it is possible that some users may decide that they could not have found the proof themselves. That does not mean, however, that the program has found something the user did not know they were looking for: clearly, by providing the agent with axioms and a hypothesis, they expected the program to find a proof. Moreover, the key concept in the proof – black and white squares – was also supplied by the user. Hence, under McCarthy's criteria, as no new concept has been introduced, the agent's solution in this scenario would not be called creative.

3.3 Scenario 3

In this scenario, the agent starts with an ability to put dominoes on a board and check whether the resulting board state covers all the squares. It is also given very limited background knowledge, in particular, it is *not* given the concept of black and white squares. It is also given an axiom about a full covering of the board, namely that the number of uncovered squares is zero. Furthermore, it is supplied with both the hypothesis that such a covering *can* be derived by putting dominoes on the board, and the hypothesis that this is not possible.

In this scenario, the agent begins by exploring the space of board states, by putting dominoes on the board and recording the board states produced. It then performs invention to produce new concepts based on the background concepts, and eventually invents the concepts of black and white squares and the concept of boards where the number of uncovered black squares equals the number of uncovered white squares. Using the evidence of the board states it has produced by placing dominoes on the board, it then induces the hypothesis that this new concept is not true of any board state possible by putting dominoes on the board. It then imagines a consequence that would follow if this induced hypothesis was true, in the following manner: it proves that, given the axiom about full coverings having no squares uncovered, the hypothesis that a covering exists is inconsistent with the induced hypothesis (i.e., it deduces the false clause).

The agent reports that it has invented the concepts of black and white squares, and the concept of board states where the number of uncovered black squares is equal to the number of uncovered white squares. It reports that it has used empirical evidence to make the hypothesis that board states gained by putting dominoes on the board never satisfy the definition of the concept. Finally, it reports the proof that, if the induced hypothesis is true, then the hypothesis that a full covering can be derived by putting dominoes on a board is false.

Despite the fact that most users would immediately realise that the induced hypothesis was true, and hence the agent had found the correct answer, we cannot say that the agent has fully solved the problem, because it did not prove the induced hypothesis. However, it seems much more likely that the user would project the word creative onto the agent, because (a) the computations involved invention, induction, imagination, deduction and proving, and it is possible that the user might conclude that he/she would not have found a similar solution and (b) the solution contains novel things that the user did not know they were looking for, namely the concept of black and white squares, the hypothesis that the number of black squares uncovered is always different to the number of white squares uncovered, and the demonstration that this disproves the hypothesis that a covering can be found by putting dominoes on the board (remember that the user provided both the hypothesis that a covering is possible and that one is not possible, hence they did not know which one was true).

3.4 Scenario 4

In this scenario, the agent starts with the same information as in scenario 3, and again explores the space of board states achievable by putting dominoes on a board, and invents the concept of black squares and the concept of white squares and the concept of board states where the number of uncovered black squares is equal to the number of uncovered white squares. It makes the empirical conjecture that no board state exists which satisfies the latter concept.

Looking at the board states before and after putting a domino on it, the agent induces the hypothesis that putting a single domino on the board covers up 1 black square and 1 white square. It then proves this fact by exhaustive experimentation. That is, it tries every possible way of putting a single domino on the board and on each occasion, it covers up a single white square and a single black square. Note that there are fewer than 64*4=256 distinct ways of putting a single domino on the board.

The induced hypothesis has therefore become a proved theorem, and from this new fact, the agent deduces that the hypothesis that the concept of board states which have an equal number of black and white states does indeed have no examples. Finally, it exploits this new fact to prove that the hypothesis that a covering can be derived by putting dominoes on a board is false, as in scenario 3. It reports to the user that it has proved that this supplied hypothesis is false.

Note that, in this scenario, the problem has been fully solved, because the falsity of the user's hypothesis has been shown, with the proof ultimately resting on the truth of an induced hypothesis which was proved via exhaustive experimentation. Moreover, it is difficult *not* to project the word creative onto the agent in this scenario. This is because (a) it has used seven different techniques associated with creativity and (b) it has invented many new, interesting things that the user did not know they were looking for. Moreover, in McCarthy's view, because it contains concepts not present in the problem formulation, the solution is creative.

4 Conclusions and Future Work

We have discussed some techniques for knowledge generation within an extended machine learning framework where the task to improve upon from experience is to find something novel and interesting that the user didn't know they were looking for. The inspiration for these techniques came from creativity studies. We believe it is important to study the plethora of terms found in the creativity literature from the bottom up. To this end, we chose a set of creativity notions, and related them to knowledge generation procedures. Using a reduced representation (logic programs), we then presented

some situations in which we believe the word could be projected onto an agent. In this way, we hope to have been fairly concrete about these creativity notions.

We presented the Mutilated Checkerboard problem as a case study, in response to McCarthy proposing this as a Drosophila for creativity studies. We presented four hypothesised scenarios where the background knowledge given to the agents, and the techniques they employed varied as they worked towards solving the problem. In three scenarios, we concluded that the problem had been fully solved, in two scenarios, we argued that the agent had acted creatively, and in only one scenario did we believe that the agent had fully solved the problem in a creative way. We came to a tentative conclusion that it was easier to project the word creative onto the agents which used more varied knowledge generation techniques (induction, deduction, imagination, etc.). We believe that the assessment of creative programs in terms of the variety of reasoning techniques they employ is worthy of further study.

Our choice of the term 'creative logic programming' is due both to the fact that the techniques involved are inspired by notions from creativity, and because, when phrased in the representation language of logic programs, the framework adds to the domain of automated logic programming (of which inductive logic programming is a sub-domain). Thus, the initial study presented here not only adds to the volume of work aimed at de-mystifying various words from the creativity literature, but also sketches a new framework for logic programming. This enables us to present the following road-map for the development of creative logic programming.

Firstly, more notions from the study of creativity need to be identified and used to inspire possible ways for an agent to add information to a knowledge base. Some possibilities, which we haven't had space to cover here, include: reflection, explanation, evaluation, interpretation, discovery, abstraction, observation, calculation, synthesis, expansion, metaphor and argumentation.

Secondly, more situations in which knowledge generation techniques associated with each of these terms need to be derived. As we have stressed, this will help us gain a more concrete understanding of the creative terms themselves, and the situations should be constructed in such a way that it is possible to argue that the creative term applies to the way in which the agent has acted. Ideally, for each of the creative notions we have introduced, including those only briefly mentioned above, there will be numerous techniques, situations and scenarios to which the word will apply.

Thirdly, in order to make the framework more concrete, we should take advantage of the formality afforded by the theory of logic programming. We originally chose logic programs as the reduced representation language because (a) we could possibly implement our ideas in Prolog (b) we could draw on the inductive, deductive and abductive techniques which are already well developed in AI and (c) there is a wealth of formal concepts and results associated with logic programs which we could draw from.

Finally, we need to identify more problems like the Mutilated Checkerboard problem. These should require a multitude of creative techniques to fully solve, and we should similarly identify scenarios wherein we can propose how an agent would solve them creatively. As another example of a mathematics problem requiring at least invention, induction and deduction, see Zeitz's plug and chug problem as discussed in [6]. Implementations will need to be made which carry out some or all of the techniques highlighted above. The implementations will need to not only solve the inspiring examples, like the Mutilated Checkerboard problem, but also solve unseen problems in such a way that the user is happy to project the word creative onto the program.

To implement a truly creative logic programming system, one possibility we aim to pursue is to extend the functionality of the HR system (which has been developed for the task of finding things the user didn't know they were looking for). This extension will be in terms of equipping HR with abilities that are currently missing from its toolbox. We have identified areas such as inductive logic programming, genetic programming, constraint solving, automated theorem proving and model generation, as possible avenues of research to draw from. At present, HR can carry out some of the techniques presented in scenario 3 above. In particular, it could be argued that the invention of the concept of board states with the same number of black and white squares uncovered is the most creative part of the solving process in scenario 3 above. After some experimentation, we have found that HR can invent this concept from very little background information. In fact, HR needs to be only supplied with two background concepts, namely the concept of the set of coordinates (squares) not covered by dominoes in a partially covered board, and the concept of the parity of a number.

The parity of an integer is taken to be 0 if it is even and 1 if it is odd. From this concept, HR invents the concept of coordinates which share the same parity, e.g., (0,0), (1,3), (0,4), etc. It then uses its negate production rule to invent the concept of coordinates with different parities. These two concepts equate to the notion of a black and a white square on the checkerboard. From these concepts, HR user the size production rule to invent the concept of the number of black coordinates (squares) not covered by a domino on a partially covered board, and does similar for the white squares. Finally, it uses it's compose production rule to invent the concept of partially covered board states which have an equal number of uncovered black and uncovered white squares. This concept turns out to have no positive examples, as all the board states have an unequal number of uncovered black and white squares. It therefore induces the non-existent conjecture that no board state is possible where this is true. Outputting this conjecture constitutes the same near-solution as that proposed in scenario 3. We hope to use this as a springboard for the production of a full solution such as that suggested in scenario 4. We find it very encouraging that HR can make the 'Eureka' step in solving this problem. In future, we hope to demonstrate that HR can fully solve this problem, given only minimal information about the domain. If it acts in a similar manner to the agent in scenario 4, using many techniques associated with creative behaviour, we will certainly argue that the program has been creative.

This paper is, we hope, the first of many exploring the notion of creative logic programming as an extension to induc-

tive logic programming. We have already begun the second stage of this study, which has been to express the way HR operates in terms of constructing a clausal (logic program) theory, using a background theory as a seed. We intend to do likewise for other machine learning and automated reasoning systems. Following this, we will endeavour to build a formal framework within which the functioning of many systems can both be described and integrated. This will enable the construction of agents which can take something like a relational database verbatim and act creatively in order to discover interesting knowledge that the user did not know they were looking for.

Acknowledgements

We would like to thank Huma Lodhi, Stephen Muggleton, Marcus Pearce, Alison Pease, Oliver Ray, and Alireza Tamaddoni Nezhad for insightful contributions to this research. We would also like to thank the organisers of the UKCRC grand challenges exercise (in particular, the participants of Panel D at the associated workshop), as this work grew out of a submission to that exercise. We are very grateful to the anonymous reviewers for their interesting and pertinent comments.

References

[1] M Abell and J Braselton. *Maple V by Example*. Associated Press Professional, 1994.

[2] M Boden. *The Creative Mind*. Weidenfeld and Nicolson, 1990.

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

[4] S Colton. An application-based comparison of automated theory formation and inductive logic programming. *Linkoping Electronic Articles in Computer and Information Science (special issue: Proceedings of Machine Intelligence 17)*, 2000.

[5] S Colton. Assessing exploratory theory formation programs. In *Proceedings of the AAAI-2000 workshop on new research directions in machine learning*, 2000.

[6] S Colton. Automated plugging and chugging. In M Kerber and M Kohlhase, editors, *Proceedings of the Eighth Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, 2000.

[7] S Colton. Automated puzzle generation. In *Proceedings of the AISB'02 Symposium on AI and Creativity in the Arts and Science*, 2002.

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

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

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

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

[12] S Colton and A Pease. Lakatos-style methods in automated reasoning. In *Proceedings of the IJCAI'03 workshop on Agents and Reasoning*, 2003 (forthcoming).

[13] S Colton, A Pease, and G Ritchie. The effect of input knowledge on creativity. In *Proceedings of the ICCBR'01 Workshop on Creative Systems*, 2001.

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

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

[16] A Kakas, R Kowalski, and F Toni. Abductive logic programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.

[17] G Kokai. GeLog - a system combining genetic algorithms with inductive logic programming. In *Proceedings of the International Conference on Computational Intelligence*, 2001.

[18] I Lakatos. *Proofs and Refutations: The logic of mathematical discovery*. Cambridge University Press, 1976.

[19] J McCarthy. Creative solutions to problems. In *Proceedings of the AISB'99 Symposium on AI and Scientific Creativity*, 1999.

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

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

[22] A Meier, V Sorge, and S. Colton. Employing theory formation to guide proof planning. In *Proceedings of the Tenth Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, LNAI 2385*. Springer, 2002.

[23] R Michalski and J Larson. Inductive inference of VL decision rules. In *Proceedings of the Workshop in Pattern-Directed Inference Systems (Published in SIGART Newsletter ACM, No. 63)*, 1977.

[24] T Mitchell. *Machine Learning*. McGraw Hill, 1997.

[25] S Muggleton. Inductive Logic Programming. *New Generation Computing*, 8(4):295–318, 1991.

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

[27] A Pease, S Colton, and A Smaill. A multi-agent approach to modelling interaction in human mathematical reasoning. In *Proceedings of Intelligent Agent Technology*, 2001.

[28] R Schapire. The boosting approach to machine learning: An overview. In *Proceedings of the MSRI Workshop on Nonlinear Estimation and Classification*, 2002.