The Effect of Input Knowledge on Creativity

Simon Colton, Alison Pease, Graeme Ritchie
Division of Informatics
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
United Kingdom
{simonco,alisonp,graeme}@dai.ed.ac.uk

Abstract

Recently, many programs have been written to perform tasks which are usually regarded as requiring creativity in humans. We can derive some commonalities between these programs in order to build further creative programs. Key to this is the derivation of certain measures which assess how creative a program is. Starting from recent proposals by Ritchie, we define possible measures which describe the extent to which a program produces novel output. We discuss how this relates to the creativity of the program.

Introduction

There has been much debate about machine creativity in artificial intelligence, cognitive science and philosophy. However, only recently have sufficient numbers of creative programs become available for us derive some commonalities between them. Such commonalities allow us to suggest ways in which creative programs can be designed, utilised and assessed. An important part of this process is the determination of a set of measures which can be used to estimate the creativity of a program.

Under certain circumstances, it may be possible to use such measures to determine whether one program is more creative than another. However, the circumstances would have to be very special, taking the design, input and output of both programs into consideration, and it is still likely that such a comparison would be deemed unfair. Similarly, it may be possible to use measures of creativity to determine whether a program is being creative at all, but this is also problematic, because creativity is such an overloaded and highly subjective word. For instance, Cohen's AARON program (Cohen 1995)[1] is cited as creative by many people, but is not thought of as creative by its author.

The worth of measures of creativity therefore lies in using them in the design of creative programs rather than the assessment of established programs. If we can agree upon a set of measures of creativity of programs, then someone writing a program can use these as a guideline for increasing the creativity of the program. If a new version of the program is assessed more favourably by some of the measures than the previous version, it is likely that progress has been made.

[1]Currently available at www.kurzweilcyberart.com

For any program which purports to be creative, an important question is the extent to which its design (including the algorithms and data it uses) is contrived to produce particular outputs. That is, has the program been 'fine-tuned' to generate specific results? Evidence of fine-tuning can affect our perception of how creative a program has been. For instance, one of the reasons that Lenat's AM program (Lenat 1982) appears creative is that it began by working in set theory, but switched to number theory, with its best results arising in the latter domain. However, our willingness to accept AM as creative is lowered by the evidence provided in (Ritchie & Hanna 1984) that AM's 'decision' to regard bags as numbers — which led it to investigate number theory — was the result of certain carefully crafted knowledge.

In order to address the question of how the knowledge input to a program affects its creativity, we first discuss the nature of the kinds of program to which our measures of creativity apply and give some examples of such programs. We then draw on Popper's philosophy of science to motivate our investigation into fine-tuning. Following this, we give an overview of Ritchie's previous work on measuring creativity (Ritchie 2001), and discuss how we will build on these. We then introduce the notion of a creative set and use this to derive some measures of fine-tuning which affect the assessment of creativity in programs. Finally, we perform a case study using the AM and HR mathematical theory formation programs (Lenat 1982), (Colton 2001).

Background

We first discuss the types of programs for which our measures will be meaningful, both by identifying properties they share and by surveying some relevant programs. Following this, we motivate our study of fine-tuning by making an analogy between creative programs and theories as described in Popper's philosophy of science.

Creative Programs

Following (Ritchie 2001), we restrict our discussion to programs which produce novel artefacts which can be qualitatively assessed by a human. The artefacts can be jokes, mathematical conjectures, poems, melodies, paintings, etc., but the restriction of qualitatively assessing them is important. To a large extent, this rules out programs such as computer algebra systems, where it is rare for the output from a calculation to be described as good, bad or anything in-between. Occasionally, the output of a calculation might be surprising, but this was not the expectation beforehand. On the other hand, with the creative systems we discuss here, the reason for running the program is to produce something which is not only novel but which can be considered of high quality. For this reason, these programs often spend much of their time internally assessing the artefacts they produced in order to prune and order their output, and direct their search.

Recent creative programs of the type we are interested in include the JAPE joke generator (Binsted 1996), the HR mathematical theory formation program (Colton, Bundy, & Walsh 1999), the MuzaCazUza melody generator (Ribeiro *et al.* 2001) and the ASPERA poetry generator (Gervás 2001). We give a brief overview of these programs below, in order to derive some commonalities not only in their output, but also in the knowledge input to them.

The JAPE program (Binsted 1996) produces simple punning riddles such as:

What do you get when you cross
a monkey and a peach?
An ape-ricot.

Although Binsted carefully does not claim that her program is creative, JAPE is the type of program that we are interested in, since its output can be judged as to its quality (as a joke), and the aim of the exercise was to have the program creating items of as high-quality as possible. Binsted tested the output in a controlled fashion by showing it to school children, who were asked to say, for each item, if it was a joke, and how funny it was. On average, the output items were deemed to be jokes, and fairly funny, though not as funny as those published in joke books for children (Binsted, Pain, & Ritchie 1997).

The HR program (Colton, Bundy, & Walsh 1999) (Colton 2001) performs theory formation in domains of pure mathematics such as group theory, number theory and graph theory. Given a little information about the domain of interest comprising some fundamental concepts in the domain (such as divisors in number theory), HR performs concept formation, conjecture making and (in algebraic domains) theorem proving and counterexample finding. The process is driven by the concept formation. That is, concepts are formed and conjectures are proposed by looking for patterns in the examples of the concepts. The nature of the concepts formed therefore dictates the nature of the conjectures.

There are 7 general production rules which turn one (or two) old concepts into a new one. As discussed later, each of these was introduced in order for HR to re-invent a classically interesting concept which it was not able to re-invent before. HR uses a set of measures of interestingness to determine which is the most interesting concept at any one time. This enables HR to build new concepts from the more interesting ones before the less interesting ones. The measures of interestingness include those which determine some intrinsic property of the concept, such as the complexity of its definition, and some relative properties of the concept, including how novel the categorisation it achieves is with respect to the other concepts. Finally, there are measures which determine whether the conjectures about a concept add to the interestingness of that concept.

The MuzaCazUza program (Ribeiro *et al.* 2001) is a melody generating program which is the successor to the SICOM program (Pereira *et al.* 1997). MuzaCazUza uses case base reasoning to generate a melody given a harmonic line over which to compose it. The cases comprise a chord, its rhythm and melody, and are taken from (at present) pieces from the Baroque period. Each case has attributes including pitch, duration of the chord and duration of the preceding and following chord.

When given a harmonic line, MuzaCazUza searches through the entire case base and scores each case by matching it with the input case. The score is taken as a weighted sum which assesses how close the input case is to the base case, using, amongst other techniques, Schöenberg's chart of the regions. The authors state in (Ribeiro *et al.* 2001) that this is fairly limited approach, which often produces obvious and less interesting solutions. They plan to give MuzaCazUza some more of the properties of SICOM, such as choosing from the best cases randomly. After the melody has been generated, there is a more interactive phase where the user can choose to apply transformations of the melody, such as mirroring certain parts of it. The user can also choose to adapt the piece using a tonal correction algorithm.

The ASPERA poetry generator also uses cased-based reasoning. ASPERA generates poetry from user-given prose which contains an intended message for the poem. It uses fragments of the text as keys for the case base which contains corresponding poem fragments. The collection of poem fragments comprises the skeleton poem and ASPERA refines this using metrical rules to adapt the best matching cases to the text provided. The adaptation takes place via an abstraction to the linguistic categories of the words in the poem fragments. The metric which performs the adaptation is chosen to be the most appropriate for the user's wishes. The user is then asked whether the resulting poem is valid. If so, then the lines of verse are analysed linguistically and incorporated into the system data files.

We see that there are some similarities between these programs. Firstly, each program produces artefacts and the quality of the artefacts is assessed either internally or with the help of a user. In HR, measures of interestingness have been derived to help it find the most interesting concepts and conjectures first. In JAPE, an external assessment of the jokes was undertaken using school children. In MuzaCazUza, after the initial generation, there is an interactive phase where the user can attempt to increase the quality of the melody by applying transformations and tonal corrections. In ASPERA, if the user validates a poem, then this information is added to the data which the program will use next time, in order to increase the quality of the output with cases from validated poems.

There is also a similarity in the input to the programs, with all the programs taking data as input and, to a large extent, having interchangeable algorithms for generation and assessment of artefacts. We address this in detail later when discussing the contributions to creativity.

Motivation from the Philosophy of Science

Following Tarski, Popper suggests that we divide the universal class of all statements into true and false, T and F, (Popper 1972). He claims that the aim of science is to discover theories (explanations) whose content covers as much of T and as little of F as possible, where the content of a theory is the set of all statements logically entailed by it. This set may also be divided into true and false statements (the theory's truth and falsity content). A good theory should suggest where to look, i.e. new observations which we had not thought of making before.

This is comparable to a situation where the universal class of all basic items in a domain is divided into good and bad, V and V'. If we describe the content of a program as its output set O which may be divided into good and bad artefacts, then we can claim that one aim of a creative program is to generate as much of V (and as little of V') as possible. A good system should suggest new areas of the search space to explore, i.e. find artefacts which we had not thought of generating before. If we accept this analogy, then Popper's criteria for evaluating theories sheds light on criteria for evaluating creative programs.

Popper sets out two criteria for a satisfactory theory (in addition to it logically entailing what it explains). Firstly it must not be *ad hoc*. That is, the theory (explicans) cannot itself be evidence for the phenomena to be explained (explicandum), or vice versa. For example if the explicandum is 'this rat is dead', then it is not enough to suggest that 'this rat ate poison' if the evidence for it having done so is it being now dead. There must be independent evidence, such as 'the rat's stomach contains rat poison'. The opposite of an *ad hoc* explanation is therefore one which is independently testable. Secondly, a theory must be rich in content. For example, a theory which explains phenomena other than the specific phenomena it was designed to explain has a much richer content, and therefore has greater value than one which is less general (the principle of universality).

Applying these criteria to creative programs, if we see a program P as the theory and the set of artefacts A we wish to generate as the phenomena to be explained, then we are interested in the independent testability of P and the richness of its content. A program which has been carefully tailored in order to produce very specific artefacts cannot be claimed to be a good program on the grounds that it produces those artefacts. There must be independent grounds for its value, such as also generating other valuable artefacts. Within the programming analogy, this is clearly connected to the richness of content criterion; the more valuable artefacts outside of A and fewer worthless artefacts a program generates, the better that program is.

It is important to note that Popper's criteria are general for all scientific theories, applying to single statement explanations as well as all-encompassing theories. They therefore apply to any program (including subsets of larger programs) able to generate artefacts in A. The conclusion of the analogy is that we should aim to make our programs as general as possible. That is, any creative program which re-invents already known artefacts should also generate a reasonable number of new, valuable, artefacts.

Contributions to Creativity

In (Ritchie 2001), some measures of program creativity were formally introduced. Ritchie suggested that judgements of creativity take into account the fact that the program designer is typically guided by some set I of (usually high-valued) artefacts called the *inspiring set*. For instance, as discussed in chapter 6 of (Colton 2001), in the development of the HR program, the concept of Abelian groups was an inspiring concept and HR was developed in order to re-invent that concept in the hope that it would also invent new concepts of a similar nature. Other inspiring artefacts for HR include the concept of self-inversing elements and prime numbers, with more details given in chapter 6 of (Colton 2001). Ritchie put forward a number of criteria for program creativity, all of which depended on a small set of factors, including I, the program's output set O and the subset V of O which was high-valued.

Ritchie suggested that, while an individual measure may not be adequate for assessing creativity alone, an overall measure of creativity could be formed by combining some of these formulae (or other similar criteria). We add to these measures by looking at the way particular knowledge may contribute to the creativity of a program. In particular, we derive measures which can be used to estimate how fine-tuned a program is. Such measures are motivated by instances of fine-tuning, such as the case with Lenat's AM described above, and also by the analogy with the philosophy of science given above.

Following Ritchie, we assume that the artefacts produced by the creative programs can be rated by humans according to their 'value' (quality). While such a subjective judgement of value is difficult in general, it is not impossible; see for example, (Binsted 1996), (Colton 2001) or (Steel 1999). Specifically, following Ritchie, we assume that a value-set, V, of high valued artefacts can be extracted from the program's output set O. The output set can be determined either by single or multiple runs, but in either case, it is taken to be those artefacts actually produced by the program, rather than those it could plausibly produce.

The degree of creativity in a program is partly determined by the number of novel items of value it produces. Therefore we are interested in the set of valuable items produced by the program which exclude those in the inspiring set I. We call this the creative set:

$$C = V - I$$

A programmer may increase the size of C by using general procedural methods as opposed to specific procedures (which are more likely to produce items in I). A general procedure might consist of dropping or negating a constraint or altering a variable (for instance Kekulé's discovery of the benzene ring, in which he negated the constraint that a string-molecule is an open curve (Boden 1992)). Such heuristics have been gleaned from examining how previous creative items were produced but are general enough to produce valuable items not yet envisaged. Therefore, the generality of the procedures in a program contributes to the degree of creativity we attribute to it.

We need a formal account of the knowledge *input* to such programs. This is not straightforward, as there is not a uniform set of items which comprise the input. Fortunately, there is often a natural classification of the kinds of information used by a program. For instance, the HR program uses a set of 7 production rules for generating new concepts, along with a set of measures of interestingness for the concepts and conjectures produced. Furthermore in HR, for each session, the user sets various parameters — mostly to dictate the form of the heuristic search — and provides a few initial concepts from which all others follow. Similarly, in AM there were 242 heuristics and 115 elementary concepts.

JAPE is primarily driven by three types of rule (schemata, description-rules, and templates), all of which assume the availability of a general purpose dictionary (in Binsted's main experiment, Wordnet (Miller *et al.* 1990) was used). Although the rules may interact, in the sense that the operation of a particular schemata may result in an intermediate structure which only certain description-rules can use, or the output of a description rule may be suitable only for certain templates, each rule has a meaning independently of the other rules.

The data given to MuzaCazUza comprises the base cases which enable it to generate melodies and the harmonic line supplied by the user. In addition to this, there are techniques for assessing how close the input case is to the base case and (in SICOM, but suggested also for MuzaCazUza) different possibilities for choosing a case from the best-matching cases (e.g. randomly). Furthermore, the user is provided with various transformations which they can apply to the melodies produced, and these transformations also make up the knowledge given to MuzaCazUza.

With ASPERA, the data supplied is also comprised of the base cases and the user input, in this case some prose which encompasses an intended meaning. Also, the metrical rules for adapting the poem fragments retrieved from the case base to the prose, and the techniques for analysing validated poems make up the knowledge given to ASPERA.

We see that the types of initial information include:

(i) procedures for generating artefacts

(ii) procedures for altering/adapting artefacts

(iii) calculations for evaluating artefacts

(iv) parameters for the search

(v) input data

We do not need to distinguish between these types here.

Some programs are designed so that, to allow experiments, certain items from the above subsets are optional. For example, the HR user can turn construction procedures and measures of interestingness on and off, change parameters and remove input data; in principle, the heuristics and initial concepts of AM include (we assume) some optional elements; JAPE's rules are independent in the sense that rules could be added or removed to alter the set of jokes generated. We therefore assume that the program can (at least theoretically) operate with different *input sets*, and this variation may affect the output.

Measures of Fine-Tuning

Let us now consider the case where input knowledge causes a program to replicate known items to a greater extent than it causes the generation of novel high-valued items; we suggest that this captures the notion of 'fine-tuning'.

We will write O_K for the set of output artefacts corresponding to input knowledge K. Then we define V_K as the set of high-valued items in O_K; R_K as the 're-inventions', i.e. the artefacts in V_K which were in the inspiring set I; and C_K as the creative set – those artefacts in V_K which were not originally in I (i.e. V_K - R_K).

If we remove a particular subset K' from K, then $V_{(K-K')}$ may differ from V_K. From this point, we shall assume a fixed set K of available input knowledge, and consider the effects of removing subsets of that overall knowledge base. We can define three convenient terms:

- K' is *creatively irrelevant* if $V_K = V_{(K-K')}$
- K' is *creatively useful* if $V_{(K-K')} \subset V_K$
- K' is *creatively destructive* if $V_K \subset V_{(K-K')}$

We are interested in the creatively destructive and especially the creatively useful subsets of knowledge. For $K' \subseteq K$ which is creatively useful, we define the *dependency set* of K', to be the set $D_{K'} = V_K - V_{(K-K')}$. This is the set of high valued artefacts which will be missing from the output if K' is removed from the knowledge set.

As above, we define $R_K = V_K \cap I$ (the re-inventions) and $C_K = V_K - I$ (the creative set). For a particular creatively useful $K' \subseteq K$, we can say that K' is *fine-tuned* if:

$$|D_{K'} \cap R_K| > 0 \text{ and } |D_{K'} \cap C_K| = 0.$$

That is, the contribution of K' to high-valued output is confined to replicating elements of the inspiring set, with no novel high-valued output being directly attributable to K'.

For cases where $|D_{K'} \cap C_K| > 0$ (i.e. there are at least some high-valued novel items contributed by K'), we can get an indication of how fine-tuned K' is by defining:

$$ft(K') = \frac{|D_{K'} \cap R_K|}{|D_{K'} \cap C_K|}$$

This returns a value greater than 1 if K' is used to rediscover more artefacts than to find new ones of value, and 1 or less otherwise. Thus it captures the notion of a piece of knowledge being introduced only in order to find particular elements of the inspiring set, with nothing else of value being lost by removing that piece of knowledge.

So far, we have a measure for the extent to which some individual subset of knowledge contributes to the fine-tuning of the whole knowledge base. In the case where $K' = K$, it follows from our definitions that a program which fails to produce any novel high-valued artefacts when using K (i.e. $|D_K \cap C_K| = 0$) but does replicate some of the inspiring set (i.e. $|D_K \cap R_K| > 0$) is deemed to be fine-tuned.

It is also interesting to consider, in the more general case, the extent to which there is fine-tuning in the various subsets of K. For this we need to be more selective about the subsets we consider. Consider the case where a subset K_1

corresponds to a dependency set D_{K_1}. That is, removing K_1 from K will result in the set D_{K_1} not forming part of the program's output. Suppose there is also a subset K_2 which is creatively irrelevant in the sense defined above, and consider $K' = K_1 \cup K_2$. If there is no interaction between K_1 and K_2, it is possible that $V_K - V_{(K-K')} = V_K - V_{(K-K_1)}$, and hence $D_{K'} = D_{K_1}$. This would mean that the larger set, K', would be rated in the same way, with respect to fine-tuning, as K_1, which is intuitively untidy. To avoid this, we introduce the following definition:

A subset $K' \subseteq K$ which is creatively useful, with dependency set $D_{K'}$, *non-redundantly contributes to* V_K if there is no subset K'' of K' such that $D_{K''} = D_{K'}$. Then in the following two measures, which both give some assessment of the extent of fine-tuning amongst the subsets of K, we restrict attention to the subsets K' which *non-redundantly* contribute to V_K. These measures describe how fine-tuned a program P is when using knowledge K (assuming P was constructed using inspiring set I):

- $m_1(P, I, K) = \frac{|K_{ft}|}{|K|}$

where $K_{ft} = \bigcup_{\{K' \subset K : K' \text{ is } fine-tuned\}} K'$

- $m_2(P, I, K) =$ maximum of $ft(K')$ over $K' \subset K$

If m_1 is greater than 0 or m_2 greater than 1, we can claim that P using K has been fine-tuned to some extent. If m_1 is 1, P using K is completely fine-tuned, in the sense that every item of knowledge in K contributes to some subset (which non-redundantly contributes to V_K) which is fine-tuned. If m_2 is greater than 1, then there is at least one such subset of K which is used more to replicate known artefacts than to find new ones of value.

Case Studies

We pointed out earlier that it is not our intention to apply our measures to established programs. Hence, as these measures are new, and no programs have been developed with them in mind, our case studies are more qualitative than quantitative.

AM is one of very few programs to have been criticised for fine-tuning in the literature (Ritchie & Hanna 1984). Ritchie and Hanna state that:

> '... it is possible to gain the impression that the successful "discovery" was the result of various specially designed pieces of information, aimed at achieving this effect.' (page 263)

This suggests that there were specialised heuristics in AM which had a disproportionate effect on its results. In fact, in (Lenat 1982) Lenat proposes this as a way for writing discovery programs:

> 'Suppose a large collection of these heuristic strategies has been assembled (e.g. by analyzing a great many discoveries, and writing down new heuristic rules whenever necessary) ... one can imagine starting from a basic core of knowledge and "running" the heuristics to generate new concepts. ... Such syntheses are precisely what AM does.' (p. 5)

This suggests that AM was written by Lenat looking at particular concepts or conjectures and adding in heuristics until AM successfully found the result. This is true of many creative programs, but unfortunately with AM, sometimes the heuristic was so fine-tuned it was introduced solely in order for AM to re-invent a single concept, e.g., the concept of number (by thinking of bags as numbers).

Returning to our measures, the situation in AM could be characterised as K containing a small (possibly unary) subset K' such that $ft(K')$ was particularly high. Hence, we could conclude that m_2 for AM would be greater than 1 and that AM was fine-tuned to a certain extent. In summary, we note that AM had more heuristics (242) than concepts it would ordinarily produce in a session (around 180).

In (Colton 2001) we went to some lengths to argue that the HR program was not fine-tuned. For instance, we pointed out that the match production rule was inspired by the concept of self-inversing elements in groups (those for which $a^{-1} = a$), but when employed in number theory, it enabled HR to re-invent the concept of square numbers. We did similarly for all the production rules, but, more importantly, we showed that all production rules were used in forming concepts which were new to us (and sometimes new to mathematics, as discussed in (Colton, Bundy, & Walsh 2000)).

We can also use the measures derived above to compare two competing knowledge subsets. Suppose a program P can be run with either with the knowledge base $K_0 \cup K_1$ or with $K_0 \cup K_2$, where K_1 and K_2 are in some sense alternative formulations that we wish to compare. Suppose that both K_1 and K_2 are creatively useful, and that the high-valued result set for the first of these is V_{K_1} and for the second knowledge base it is V_{K_2}. If $V_{K_1} \subset V_{K_2}$, then it follows that $D_{K_1} \subset D_{K_2}$ and also $C_{K_1} \subseteq C_{K_2}$. Hence $|D_{K_1} \cap C_{K_1}| \leq |D_{K_2} \cap C_{K_2}|$, so $ft(K_1) \geq ft(K_2)$. That is, the input knowledge variant K_2 which corresponds to the larger V_K will at worst be as fine-tuned as the other input knowledge K_1. If $C_{K_1} \subset C_{K_2}$, K_2 will be less fine-tuned.

Such a situation occured during the development of HR — as discussed in chapter 6 of (Colton 2001) — when two production rules in HR were replaced by a new, more general, rule. In particular, the conjunct rule was employed to take two concepts with, for example, definitions $p_1(a, b, c)$ and $p_2(a, b, c)$ and produce a concept with definition $p_1(a, b, c) \wedge p_2(a, b, c)$ [where p_1 and p_2 are predicates defining some relation over objects a, b and c]. The common rule was designed to take a single concept with definition $p(a, b)$ and produce a concept with a definition $p(a, b) \wedge p(a, c)$. When writing a new rule, the compose rule, which took two functions $f(a, b)$ and $g(a, b)$ and produced the concept $f(g(a, b))$, we realised that this could be written in such a way as to incorporate the functionality of the previous two also. The details of this are given in (Colton 2001), but are not relevant here. In addition to making HR more general and more comprehensible, the new rule covered all of the output from the previous two and more. So, in this case, the generalisation of the two previous rules *increased* C_K for HR so the new version was less fine-tuned than the previous version. We hope to perform a more detailed examination of this case study in future.

Conclusions and Further Work

Of course, there is definitely a need for simulations, whereby a program is written to re-produce discoveries made by humans. For example, the BACON programs (Langley *et al.* 1987) were written to provide plausible ways for a computer to re-discover certain laws from the physical sciences. Such simulations often highlight general ways to proceed and more creative programs are built as a result. However, we believe that fine-tuning in programs purporting to be creative needs to be addressed. We have sketched a formalisation of what it means for a creative program to be fine-tuned and we have noted that the more fine-tuned a program is, the less creativity we attribute to it. This contributes to the question of which processes can be deemed creative and so enhances the approach to estimating the creativity of a program based on its input and output. We believe this approach is very important in the study of machine creativity.

Our approach may be applied to existing programs whose set of inspiring items I is possible to identify. This would then support or undermine claims of creativity. However, since I may be difficult to identify in retrospect (other than the set of all artefacts known to the programmer) the main value of our work lies in its role as a guideline for researchers currently writing creative programs (who can record their inspiring set). In particular, such measures of creativity are useful when writing improved versions of creative programs, as in the illustrative example above with HR.

More work is required on these measures. In particular, these and some of the measures in (Ritchie 2001) depend on the notion of an inspiring set, which may not apply completely to certain creative programs. For example, Cohen may have improved the AARON program so that it could draw trees, but the artefacts it produces are not just drawings of trees, but of scenes with humans and trees in them, and there is no particular drawing of a tree which acts as an inspiring artefact. In this case, we would have to relax our definition of inspiring set to include parts of artefacts rather than entire artefacts. Similarly, to make these measures more general, it is likely that we will need to adopt a more fuzzy notion of good and bad artefacts.

The assessment of machine creativity is beginning to be recognised as an important area of this field, with general guidelines such as those in (Pease, Winterstein, & Colton 2001) required as well as concrete measures, such as those presented in (Ritchie 2001) and those derived here. Such measures are imperative if the study of machine creativity is to become a formal research programme. Furthermore, we hope that such notions will be very useful for researchers writing creative programs in the future.

Acknowledgments

This work is supported by EPSRC grants GR/M98012 and GR/M45030. The first author is also affiliated with the Department of Computer Science, University of York. We would like to thank Alan Smaill for some important input to this work and the anonymous reviewers for their comments on an earlier draft of this paper.

References

Binsted, K.; Pain, H.; and Ritchie, G. 1997. Children's evaluation of computer-generated punning riddles. *Pragmatics and Cognition* 5:2:309–358.

Binsted, K. 1996. *Machine Humour: An Implemented Model of Puns*. Ph.D. Dissertation, Department of Artificial Intelligence, University of Edinburgh.

Boden, M, A. 1992. *The Creative Mind*. Abacus.

Cohen, H. 1995. The further exploits of AARON, painter. *Stanford Electronic Humanities Review* 4:2.

Colton, S.; Bundy, A.; and Walsh, T. 1999. HR: Automatic concept formation in pure mathematics. In *Proceedings of the 16th IJCAI*, 786–791.

Colton, S.; Bundy, A.; and Walsh, T. 2000. Automatic invention of integer sequences. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, 558–563.

Colton, S. 2001. *Automated Theory Formation in Pure Mathematics*. Ph.D. Dissertation, Division of Informatics, University of Edinburgh.

Gervás, P. 2001. Generating poetry from a prose text: Creativity versus faithfulness. In Wiggins, G., ed., *Proceedings of the AISB'01 Symposium on Artificial Intelligence and Creativity in Arts and Science*, 93–99.

Langley, P.; Simon, H.; Bradshaw, G.; and Żytkow, J. 1987. *Scientific Discovery - Computational Explorations of the Creative Processes*. MIT Press.

Lenat, D. 1982. AM: Discovery in mathematics as heuristic search. In Lenat, D., and Davis, R., eds., *Knowledge-Based Systems in Artificial Intelligence*. McGraw-Hill Advanced Computer Science Series.

Miller, G.; Beckwith, R.; Fellbaum, C.; Gross, D.; Miller, K.; and Tengi, R. 1990. Five papers on wordnet. *International Journal of Lexicography* 3:4. Revised March 1993.

Pease, A.; Winterstein, D.; and Colton, S. 2001. Evaluating machine creativity. In *Workshop on Creative Systems, 4th International Conference on Case Based Reasoning*.

Pereira, F.; Grilo, C.; Macedo, L.; and Cardoso, A. 1997. Composing music with CBR. In *First International Conference on Computational Models of Creative Cognition, Dublin, MIND-II*.

Popper, K. 1972. *Objective Knowledge*. OUP.

Ribeiro, P.; Pereira, F. C.; Ferrand, M.; and Cardoso, A. 2001. Case-based melody generation with MuzaCazUza. In Wiggins, G., ed., *Proceedings of the AISB'01 Symposium on Artificial Intelligence and Creativity in Arts and Science*, 67–74.

Ritchie, G., and Hanna, F. 1984. AM: A case study in methodology. *Artificial Intelligence* 23:249–268.

Ritchie, G. 2001. Assessing creativity. In Wiggins, G., ed., *Proceedings of the AISB'01 Symposium on Artificial Intelligence and Creativity in Arts and Science*, 3–11.

Steel, G. 1999. Cross domain concept formation using HR. Master's thesis, Division of Informatics, University of Edinburgh.