On the Nature of Mathematical Judgement: Reply to Penrose

Alan Bundy

April 26, 2004

1 The Penrose Argument

In his book, 'The Emperor's New Mind', [Penrose 89], Roger Penrose mounts a logic-based attack, not just on artificial intelligence in general, but on automatic theorem proving in particular. My first reaction was that this was just a repetition of the Lucas 'Gödel Theorem' argument, but it is more subtle than that, and deserves a separate refutation. I am grateful to Frank van Harmelen for explaining the difference to me.

Essentially, the Penrose argument is as follows (see chapter "The non-algorithmic nature of mathematical insight" in [Penrose 89]).

> Consider the procedure by which mathematicians decide whether a given proposition is true or false. Let us call this the *judgement procedure*[1]. The judgement procedure is universally shared by mathematicians, since an argument which convinces one mathematician will convince all of them. The steps in these arguments are not obscure or complicated, since good mathematical arguments are characterised by steps which are simple and obvious. However, Gödel's Incompleteness Theorem shows us that the judgement procedure cannot be an algorithm[2]. For suppose it was, then it could be implemented as theorem proving in a formal logical theory. By Gödel's Theorem, there is a formula of this theory that is true but unprovable, *i.e.* there is a formula that every mathematician would accept as true, but which is not judged to be true by the procedure that every mathematician is supposed to use to judge truth. Contradiction - QED. Since the judgement procedure is non-algorithmic it cannot be implemented in

[1]This is my terminology. Although it plays a central role in his argument, Penrose does not have a name for this procedure.

[2]By which he appears to mean a decision procedure, although a semi-decision procedure is sufficient for the purposes of the argument.

> a computer program. However, it is implemented in human mathematicians. Therefore, human intelligence has some property that computer programs cannot have.

This argument turns around a systematic ambiguity about just what the judgement procedure does, or rather what its input is. Once this ambiguity is exposed the argument falls apart. Nevertheless, the analysis does suggest a way in which automatic theorem proving programs might be significantly extended. I will take up these two points in turn.

2 An Algorithm for Mathematical Judgement

What does Penrose mean by the judgement procedure? From the form of the Penrose argument I take him to mean a procedure for checking the correctness of a proof or any other argument advanced as to the truth of a proposition, rather than, say, a procedure for finding such proofs, or for checking the truth of a formula without the aid of a proof. It would be very surprising if this procedure were not algorithmic, since it is one of the simplest algorithms that we automatic theorem proving researchers are called on to write. We call such procedures *proof checkers*.

The natural way to write a general purpose proof checker is as a program of three inputs, say $prf\_chk(Thm, Prf, Fml\_Sys)$ which returns 'yes' or 'no' according as alleged proof, Prf, is or isn't a proof of the formula, Thm, in the formal system, Fml_Sys. This formal system input, Fml_Sys, is overlooked in the Penrose argument. It will consist of a description of the axioms and rules of inference that constitute the formal system. The proof checking procedure checks that each step of the alleged proof, Prf, is either an axiom in Fml_Sys or follows from earlier steps by a rule of inference in Fml_Sys. It also checks that the last step of the proof is, Thm, the formula whose proof is being checked.

How would such a proof checker fare on some Gödel proposition $P_k(k)$ that is true of arithmetic, but not provable in a particular formalisation of arithmetic, say *arith*? This would depend on the formal system input to the program. If it were *arith* then, whatever alleged proof we input to the program, the proof checker would deny that this was a proof of the proposition $P_k(k)$, *i.e.* $prf\_chk(P_k(k), Prf, arith)$ would return 'no' for all Prf. However, if we formalised and input both the meta-theory within which this version of Gödel's Theorem was expressed, say *meta*, and the informal argument used to establish the truth of $P_k(k)$ in this meta-theory, say *gödel*, then the proof checker would confirm that $P_k(k)$ was provable in this meta-theory, *i.e.* $prf\_chk(P_k(k), gödel, meta)$ would return 'yes'. Of course, we could then use Gödel numbering to construct a new proposition, $P'_k(k)$ which was true, but not provable in *meta* and repeat the above exercise indefinitely.

So we see that Penrose's argument collapses. His judgement procedure is perfectly algorithmic. Indeed, it is a decision procedure. Mathematicians and prf_chk both judge the formulae $P_k(k)$ to be true using the proof *gödel* in formal system *meta*. Mathematicians and prf_chk both judge the formulae $P_k(k)$ to be unprovable in the formal system *arith*. This does not call for non-algorithmic reasoning.

What Gödel's Theorem really demonstrates, is not the non-algorithmic nature of proof checking, but the inadequacies of our attempts to formalise mathematical knowledge. We will always leave out some method of proof, *i.e.* some axiom or rule of inference, that mathematicians might consider acceptable. In the example of arithmetic we are limited by our formalisation of induction. However we try to formalise it, there will be some kinds of induction that escape the formalisation. This is essentially the message of the Kirby/Paris proof about the termination of the Goodstein function, [Kirby & Paris 82]. They show that the Goodstein function can be proved to terminate using ϵ_0 induction, but cannot be proved to terminate by induction over any lesser ordinal. Peano arithmetic only allows induction over ordinals less than ϵ_0. If we added ϵ_0 induction to Peano arithmetic, then there would still be other true but unprovable formulae.

3 The Representation of Proof Steps

Perhaps we can reformulate Penrose's argument to be a critique of our ability to represent proof steps as axioms and rules of inference in a logical formalisation and hence in a computer implementation. The point would be that mathematicians agree upon the nature of mathematical proof, but that it is impossible to represent this agreement as a formal system. Hence it is impossible to build a computer program which acts as an 'artificial mathematician', since it cannot be told everything that mathematicians know about the nature of proofs.

However, mathematicians do *not* agree on the nature of mathematical proof, so this argument is wrong too. There is a long history of argument among mathematicians as to the acceptability of both axioms and rules of inference. The earliest example I can think of is the dispute among the Pythagoreans over the alleged proof of the non-rational nature of $\sqrt{2}$. Then there was the long debate over the acceptability of the fifth Euclidean postulate leading to the controversy over the non-Euclidean geometries. With the advent of the classical/intuitionist debate, even the form of the underlying logic was brought into dispute.

One of the most revealing incidents was the discovery of Russell's paradox. Russell's form of argument was considered unexceptional before he showed that it led to a contradiction. The reaction was a deeper analysis of the legality of different proof steps and a variety of different proposals as to ways of limiting them to avoid contradiction. This incident showed that, if one wanted to avoid contradiction, then one did not have complete freedom to choose one's proof steps

— even apparently self-evidently true steps might turn out to be suspect on closer analysis.

So if humans argue over the legitimacy of different proof steps, we should not expect a neat algorithm for determining whether mathematical judgements are correct, *independent* of the formal system in which they are claimed to hold. This demonstrates the necessity of the third, formal system, argument to prf_chk.

4 Extended Theorem Provers

Thus to emulate a human mathematician with a computer program we want to build a system that can not only prove theorems using a particular formal system, but can reason *in* many different formal systems, can reason *about* different formal systems and can develop new formal systems.

The first step in building extended theorem provers is to give them the ability to be told new axioms and rules of inference. Even the simplest resolution theorem provers already have this ability, at least as far as the axioms are concerned. Logical framework systems (*e.g.* [Harper *et al.* 87]) do this in a more sophisticated way by being capable of being told new underlying logical formalisms and hence new rules of inference. More sophisticated systems might try to propose new axioms and rules for themselves. For instance, new axioms could be generalised from example formulae, *e.g.* using Plotkin's technique, [Plotkin 69, Plotkin 71]. His program learnt the group theory axioms from properties of examples of groups. Alternatively, one could build a system which used Gödel numbering to construct true but unprovable formulae and then adopted them as new axioms. Or new rules of inference could be extracted by explanation based learning from examples of formula sequences which some external agency alleged to be proofs.

The next step is to give an extended automatic theorem prover the ability to reason about formal systems. For instance, to discover a formal system to be incomplete or inconsistent and to modify it accordingly. It might add, delete or amend both axioms and rules of inference, including changing the underlying logic. These modifications might also apply to the meta-logic it was using to reason about formal systems. It might compare notes with other similar programs or with human mathematicians. In this and other ways it would evolve and develop a variety of formal systems and prove theorems in them.

Suppose we take such an extended automatic theorem proving program to be our artificial mathematician and apply the Penrose argument to it. First note that there are lots of possible varieties of such programs, but they will each have a very similar proof checking sub-program. Thus they will tend to agree about whether a particular proof proves a particular theorem in a particular formal system, but they will tend to differ on which formal systems they evolve and then work in. Just like human mathematicians, it will be useful if they communicate to criticise each other's formal systems. This mutual criticism will help them avoid inconsistent formal systems and concentrate their efforts on systems of common interest. This critical function will make them even more sophisticated. I have no idea how or whether such extended theorem proving systems could be implemented, but this will not matter for the remaining argument.

Such an extended automatic theorem proving program could not be a decision procedure. Since it will contain a complete theorem prover, it will be at best a semi-decision procedure. This will not effect the Penrose argument in that it will still be possible to implement it as theorem proving in some formal system. But what formal system? It will not be some standard formal system. For instance, if we implemented an extended automatic theorem proving program for arithmetic as a formal system, the implementing formal system would not be our formalisation of arithmetic, *arith* (although it might contain *arith* as a subsystem). It would be a huge, messy, meta-level formal system, $\mathcal{F}$ say. The easiest way to see this is to imagine a Prolog implementation. The clauses of this program are approximately[3] the axioms of $\mathcal{F}$. Moreover, as this program evolved over time $\mathcal{F}$ would change. (It would assert new clauses into itself.) Let $\mathcal{F}(t)$ be the formal system at time t.

At any time we could prove a version of Gödel's Incompleteness Theorem for $\mathcal{F}_t$. This proof would identify a formula, $\mathcal{P}_t$, that was true but unprovable in $\mathcal{F}_t$. Some more sophisticated program might already have proved $\mathcal{P}_t$, or it might be proved in $\mathcal{F}'_t$, where $t' > t$, by some future variant of our program. Of course, $\mathcal{P}'_t$ would be true, but unprovable in $\mathcal{F}'_t$. So the inability to prove a particular true formula is only a temporary indisposition, whereas the inability to prove *some* true formula is a permanent one.

It is not clear to me what might be the significance of these Gödel formulae of $\mathcal{F}_t$. In Gödel's Second Incompleteness Theorem he arranged things such that the Gödel formula encoded the consistency of the formal system. Thus a program would be unable to prove itself consistent. So what? Even if it was consistent this would not stop it proving theorems in an inconsistent object-level formal system and finding contradictions. Even if it was inconsistent it might never uncover a contradiction in normal working and it is not clear what it would mean if it did. For instance, suppose the proof checking and theorem proving sub-systems were consistent, but the learning sub-system was inconsistent. Would it matter that some daft axioms were suggested?

Note that human reasoning is also incomplete in this same sense. We humans cannot be sure that our total reasoning systems are consistent. The discovery of Russell's paradox taught us all humility in this respect. Note that in reaching this conclusion we do not have to assume that reasoning is fallible because it is probabilistic or heuristic, as Dennett does in his review of Penrose's book in [Dennett 89]. These limitations will hold even for deductive systems.

[3]The impurities of Prolog cause the 'approximately' caveat.

5 Conclusion

I have tried to analyse the Penrose argument. This analysis reveals a systematic ambiguity over the procedure for forming mathematical judgements. A simple version of this judgement procedure would be a proof checker that took as input the formal system within which a proof was to be checked. An algorithmic proof checker could easily be built. No violation of Gödel's Incompleteness Theorem is involved. Hence, the Penrose argument is faulty. There is no special limitation of artificial reasoning over human reasoning.

We see that the Penrose argument metamorphoses into an exposure of the inadequacies of formal representations of methods of mathematical proof. To cope with this humans have learnt to develop and evolve these formal representations and to compare and contrast them. Contrary to Penrose's claim, there *is* disagreement about the acceptability of certain methods of proof, and hence disagreement about which is the best formal representation to adopt.

This suggests that those of us building artificial reasoning systems should also build what I have called extended theorem provers that evolve and compare their methods of reasoning. As well as proving theorems these would also conjecture new formal systems within which to prove them. Extended theorem provers would inevitably be non-algorithmic, but so are humans on this extended task of mathematical judgement.

References

[Dennett 89] D. C. Dennett. Murmurs in the cathedral. *Times Literary Supplement*, Sept 29 - Oct 5:1055–1056, 1989.

[Harper *et al.* 87] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proc. of the Second Symposium on Logic in Computer Science*, 1987.

[Kirby & Paris 82] L. A.S. Kirby and J. Paris. Accessible independence results for Peano Arithmetic. *Bull. London Math. Soc.*, 14:285–293, 1982.

[Penrose 89] R. Penrose. *The Emperor's New Mind*. Vintage, 1989.

[Plotkin 69] G. Plotkin. A note on inductive generalization. In D. Michie and B. Meltzer, editors, *Machine Intelligence 5*, pages 153–164. Edinburgh University Press, 1969.

[Plotkin 71] G. Plotkin. A further note on inductive generalization. In D Michie and B Meltzer, editors, *Machine Intelligence 6*, pages 101–126. Edinburgh University Press, 1971.