The Paradox of the Case Study

Alan Bundy

September 2004

1 Introduction

In order to demonstrate the scalability of automated reasoning techniques, it is important to embark on large-scale case studies. As a field, we need to present a reward system for the completion of such case studies. However, there is a paradox at the heart of the case study. It is this paradox, and its resolution, that I intend to explore in this note.

The classic example of the theorem proving case study is Shankar's proof of Godel's incompleteness proof using the Nqthm theorem prover [Shankar, 1994]. Shankar's reward for this case study was that he received a PhD. Note that the theorem prover was developed by third parties: Shankar's supervisors Bob Boyer and J Moore.

2 The Paradox

It is my contention that any good piece of scientific or engineering research develops a hypothesis and then provides evidence to support (or refute) that hypothesis. What is the hypothesis in the case of a theorem-proving case study? The obvious hypothesis, is that the theorem prover is capable of proving hard theorems. The evidence supporting this hypothesis is the automated proof of a hard theorem.

What would be the ideal such evidence? Presumably, that using the most obvious and direct representation of the theory and the theorem, that the theorem prover could prove the theorem totally automatically and within minutes if not seconds. Herein lies the paradox. If it was that easy, then this work would scarcely be worthy of a PhD. However, if the work was so hard that it *was* worthy of a PhD, then the evidence supporting the hypothesis would be rather thin. If the student had had to work really hard to find an appropriate representation of the theory and the theorem; if s/he had had to work even harder to search a huge search space, guiding the prover at nearly every step, then the prover would have been shown to be barely capable of proving the hard theorem, and then only under the direction of an expert in the field. It seems that the student's success is inversely proportional to the success of the theorem prover. This is the paradox of the case study. What industry wants

is push-button and fast technology, but it seems that we are unable to reward case studies that demonstrate such technology[1].

Of course, this analysis is superficial. Any theorem prover has a range of application: from simple theorems that it can prove totally automatically, to hard ones at the limit of its capacity, requiring expert guidance over multiple steps. The case study typically explores the outer limits of this range: those theorems that are barely reachable and then only under expert guidance. So a more for sophisticated hypothesis is that the outer limits of the theorem prover under test exceeds that of rival theorem provers. But then, of course, the evidence should include unsuccessful experiments to prove the same theorem using each of these rival provers. This is seldom demonstrated. Indeed, although the experimenter will usually not attempt any case studies using rival provers, the champions of many rival provers frequently take up the challenge, subsequently showing that their provers are equally, if not more, capable of completing the case study. This refutation of the implicit hypothesis often occurs *after* the original student has been rewarded with a PhD.

A further problem with this kind of case study is it becomes an awful slog. The student has to put his/her head down and push on relentlessly for weeks and months, even years, overcoming one difficulty after another, until they finally produce a huge proof, which no one else may ever study in detail. It is easy to get disheartened, to wonder whether it is all worth it, or what the point is. The attrition rate for case study students is much higher than for other kinds of project.

3 Can we Find Another Hypothesis?

One way out of this paradox, is to seek an alternative hypothesis to those presented in the last section. I propose that the case study be seen as an *analysis* of the system under test. We are asking not just whether the system can prove this hard problem, but how *easy* the prover makes producing this proof: what kind of tools does it provide to support the representation of the theory or the theorem; what kind of search-control tools help the user guide the proof; what kind of visualisation tools help the user to understand the structure of proof; what kind of analysis tools help the user to identify problems and solve them. The student might also try to classify the types of intervention that were required to guide the proof. For instance, was it enough to set the parameters of existing heuristics or tactics, was it necessary to write new kind of tactics, or was a major overhaul of the theorem prover necessary? In such an analysis, a large part of the work will be in the *development* of an appropriate hypothesis.

As an exemplar of what I have in mind, I will describe the kind of analysis that we did in Francisco Cantu's PhD project, in which he used our Clam inductive proof planner to verify various hardware algorithms and systems

[1]Unless the case studies are carried out by the system's developers. A short evaluation section within a system description *would* work.

[Cantu *et al*, 1996]. He kept a record of the kind of intervention that was necessary to guide the proofs. He classified these interventions by the kind of person who would be needed to make such interventions. For instance, choices between existing tactics might be made by the hardware developer within a user company without any deep knowledge of the theorem prover. However, if new tactics were required, then it might be necessary to call in the company's local expert. Finally, if a major overhaul of the prover was needed, then it would probably be necessary to ask the prover's supplier to issue a new version. Francisco tried to demonstrate that most interventions were of a simple nature that could be handled by the user or, at least, within the user's company, and did not require supplier intervention. He further demonstrated that the interventions were front-loaded and lumpy. That is to say, most interventions were required at the beginning of a series of experiments on a new class of algorithms or systems. After tuning the prover to this new class, most subsequent proofs required little or no human guidance. In this way, he was able to develop a usage scenario for the prover, showing that it could be a practical tool within a company doing hardware development.

This kind of analysis is both more sophisticated and more interesting than testing the simple hypotheses that this prover can prove hard theorems, or harder ones than its rivals. It avoids the case study paradox since the effort required from the student is now directly proportional to the value of the analysis that is produced from the case study. Moreover, it is much more interesting work, since the student must continually pay attention not only to the course of the proof, but to the experience of proving and to the assistance provided (or not provided) by the theorem prover. It is this analysis, rather than the proof, that is the product of the case study. This is much more likely to be useful to the developers of the theorem prover and to the user community, so, unlike the gory details of the proof itself, the analysis will be widely disseminated and appreciated.

4 Conclusion

In this note, we have argued for a more sophisticated hypothesis to underlie case study research. The simplistic hypothesis that the prover can prove hard theorems — or harder ones than its rivals — leads to a paradox in which the success of the student is inversely proportional to the success of the prover, and where the proof process becomes a relentless and unrewarding slog. The more sophisticated hypothesis consists of an analysis of the strengths and weaknesses of the prover. It consists of a series of small hypotheses: the prover provides good visualisation or analysis tools, the prover provides good automatic search control guidance, etc. It leads to more interesting research because the student must demonstrate awareness at multiple levels simultaneously: at the object-level of guiding the search for a proof and at the meta-level of the experience of the proof process. It also leads to research that will be more useful to both the developers of the prover and their user community.

Superficially, this discussion seems relevant only to the interactive theorem-proving community. This is not correct. Case studies are also carried out within the totally automated theorem proving community, for instance, EQP's proof of the Robbins Algebra Conjecture [McCune, 1997]. Note that this proof was not produced from a single run of the prover on the problem. Rather, it took place over a decade in which: the proof was divided up into lemmas, the theorem prover was developed and refined, and a series experiments was tried with a wide variety different parameter settings. The kind of analysis proposed above is just as relevant to this kind of case study as to those using explicitly interactive provers. Indeed, we badly need such analytical case studies of automated provers, to better understand and improve their usability.

References

[Cantu *et al*, 1996] Cantu, Francisco, Bundy, Alan, Smaill, Alan and Basin, David. (1996). Experiments in automating hardware verification using inductive proof planning. In Srivas, M. and Camilleri, A., (eds.), *Proceedings of the Formal Methods for Computer-Aided Design Conference*, number 1166 in Lecture Notes in Computer Science, pages 94–108. Springer-Verlag.

[McCune, 1997] McCune, W. (1997). Solution of the Robbins problem. *J. Automated Reasoning*, 19(3):263–276.

[Shankar, 1994] Shankar, N. (1994). *Metamathematics, Machines, and Gödel's proof.* Cambridge University Press.