On the Difficulty of Automating Inductive Reasoning

Robert S. Boyer and J Strother Moore

Computer Sciences Dept., University of Texas, Austin, Texas 78712
and Computational Logic, Inc., Austin, Texas

Inductive reasoning is basic to computer science, proof theory, and number theory. So why has there been so little work on induction in automated reasoning, at least in comparison to resolution or rewriting?

One explanation is that many researchers in program verification mistakenly devoted most of their inductive research efforts to implementing 'verification condition generators', which were implicit, ad hoc, programming-language specific, induction principles. This misplaced emphasis was perhaps based upon the mistaken notion that a verification condition generator was all the induction one needed to prove programs correct.

Another explanation is that most theorem-proving work has been 'first order,' but a typical first order axiomatization of induction involves an axiom schema whose naive realization would be as an infinite number of clauses, too many to type in. Or perhaps it was that the 'skolemization' that would be required to add such an infinite number of first order axioms is too hideous to contemplate, much less turn into clauses.

A more basic issue is that of search strategy. Which instances of the induction schema should one actively consider? This question is especially important and hard for resolution style theorem-proving techniques, which may start work on a simple enough conjecture but will typically build up, in the course of a proof, a much larger working set of clauses. There does not seem to be a known good answer to the question: Which of the many clauses being currently retained should one use as 'the formula to induct on'?

Probably the most important explanation of the difficulty of induction is that the problem is merely an instance of the more general problem of dealing with schematic formulas. In some sense, the entirety of mathematics is encapsulated in the (unfortunately unsound) single, little axiom of set theory that Frege might have written in this way: $x \in \{x : p(x)\} \leftrightarrow p(x)$. When dealing with such a schematic axiom, how can a prover sensibly guess which instances of p to consider? Without a really good way to answer such questions, one meets with the futility of the British Museum Algorithm. For those too young to know, the British Museum Theorem-Proving Algorithm enumerates, mindlessly but systematically, all Hilbert style proofs, until it finds a proof of the given theorem, as though visiting the British Museum, where one gets to see at least one example of everything.

The success of the Boyer-Moore theorem prover at 'guessing the right induction' probably stems from the fact that the guessing is driven off of the 'dual' of the recursive definitions in the 'current conjecture,' and the fact that the theorem prover was carefully designed so as to make the notion of the 'current conjecture' reasonably clear, even after induction, rewriting, case-splitting, generalization, etc. One measure of the success of this approach is that in running the current Boyer-Moore test suite, which consists of about 16,000 theorems, crafted by about 16 people over 20 years, the 'right induction' is apparently chosen over 3,800 times, whereas explicit user choice of which induction to do occurs only about 1,000 times. Theorems proved include the correctness of a manufactured chip and a compiler targeting the chip, number theory through quadratic reciprocity, and proof theory, including Gödel's incompleteness theorem and the Church-Rosser theorem.