Using Implicit Induction to Guide a Parallel Search for Inconsistency

Graham Steel, Alan Bundy and Ewen Denney
Division of Informatics
University of Edinburgh
{grahams,bundy,ewd}@dai.ed.ac.uk
http://www.dai.ed.ac.uk/˜grahams

February 21, 2002

Abstract

We describe the first full implementation of the Comon-Nieuwenhuis method for implicit induction, including a consistency checker, in a novel system where the proof and refutation programs communicate via sockets. This allows the system to attempt to prove and disprove a conjecture at the same time, using parallel theorem proving processes. As well as refuting several non-theorems, this system has accomplished what is, to the best of our knowledge, the first fully automated proof by implicit induction of the commutativity of gcd. This had been posed as a challenge problem to the technique in the past.

1 Overview of The System

Comon and Nieuwenhuis described the operation of a refutation complete proving system based on an implicit induction strategy in their paper, [1], but their implementation was not complete. We have produced a complete implementation, described in figure 1. The input to the system is an inductive problem in `Saturate` format and an I-Axiomatisation (as defined by Comon and Nieuwenhuis, see [1]). The version of `Saturate` customised by Nieuwenhuis for implicit induction (the rightmost box in the diagram) gets the problem file only, and proceeds to pursue inductive completion. Every non-redundant clause generated is passed via the server to the refutation control program (the leftmost box). For every new clause generated, this program generates a problem file containing the I-Axiomatisation and the new clause, and spawns a standard version of `Saturate` to attempt to find an inconsistency in the file. Crucially, these spawned `Saturates` are not given the original axioms. This means that most of the search for an inconsistency is done by the prover customised for inductive problems and the spawned `Saturates` are just used to check for inconsistencies between the spawned clauses and the I-Axiomatisation. This should enable an inconsistency to be found more quickly than if we just gave the axioms, the I-Axioms and the conjecture to a first-order prover.

It is necessary to spawn the standard `Saturate` processes in parallel to the implicit induction process because inductive completion may not terminate. A concurrent check for consistency allows us to detect incorrect conjectures even in this case. For example, the system has refuted $gcd(x, x) = 0$, a problem for which the completion process is non-terminating. If a refutation is found by a spawned prover, the proof is written to a file and the completion process and all the other spawned `Saturate` processes are killed. If completion is reached by the induction prover, this is communicated to the refutation control program, which will then wait for the results from the spawned processes. If they all terminate with saturation, then there are no inconsistencies, and so the theorem has been proved.

All spawned `Saturate` processes output to `stdout`, but a separate `xterm` window is launched for the implicit induction prover. This is because some user interaction is sometimes required for the implicit induction process; if it was connected to the main stream, it would accept the output from another process as its input. The user interaction required is the selection of the innermost defined symbol in order to determine an inductively complete position, [1], taking advantage of the fact that our problems are in constructor theories to reduce the

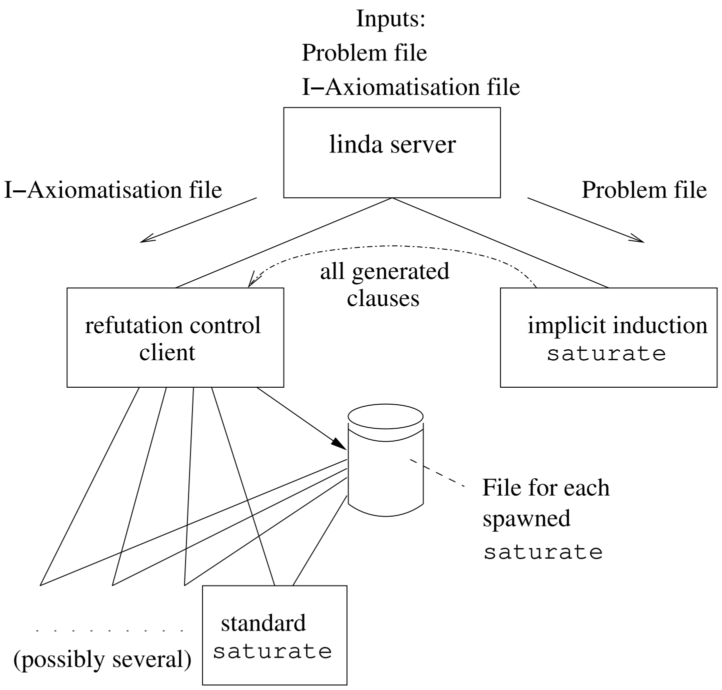

Figure 1: System Operation

amount of search required. This will be automated in the future. When the theorem proving process is over, the user also has to type the commands required to recover the proof.

2 Examples

The system has proved a small number of theorems about the *gcd* function, including commutativity and idempotence, as well as theorems about even numbers and the $>$ predicate. Several non-theorems involving the same function symbols have also been detected. For a table of results and some sample output see `http://www.dai.ed.ac.uk/~grahams/linda/arw.html`.

3 Further Work

Our eventual aim is to use this system (or one of its successors) for investigating conjectures about security protocols, and then to automatically extract attacks from the refutations. The idea is to produce a first-order version of the inductive formalism used by Paulson, [2], allowing us to reason about protocols involving an arbitrary number of agents. The main disadvantage of Paulson's method is that considerable experience with the Isabelle prover is required in order to analyse a new protocol. This is especially true if the proof attempt breaks down, leaving the user unsure as to whether there is a problem with the proof attempt, e.g. another lemma or a generalisation is needed, or whether the protocol itself is faulty. A system which could automatically synthesize attacks in the case of a faulty protocol would alleviate this problem.

References

[1] H. Comon and R. Nieuwenhuis. Induction = I-Axiomatization + First-Order Consistency. *Information and Computation*, 159(1-2):151–186, May/June 2000.

[2] L.C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6:85–128, 1998.