Intelligent Theorem Proving for Specific Domains

Paulo Jorge de Oliveira Cantante de Matos

Instituto Superior Técnico, pocm@mega.ist.utl.pt

1 Introduction

To the one who develops a theorem prover there are two possible approaches. One can try to understand what people do when they create proofs and to write the theorem prover emulating that process or to make use of the systematic work of the logicians in reducing logical reasoning to standard canonical forms on which algorithms could be based. Both approaches have many obstacles but it seems that the most successful developments incorporates insights deriving from both approaches[1].

It seems however that research has followed the latter approach. Although Artificial Intelligence techniques have been neglected in Automated Deduction there are already some research in learning previous proof experience[2] and proof planning[3]. In a Larry Wos interview he refers to the use of Artificial Intelligence in Automated Reasoning[4]:

"But, he [Wos] conjectures, there may soon be hand-held 'thought processors' using automated reasoning to help humans avoid such logistical errors as scheduling two meetings at the same time, or designing a building but forgetting to add waste disposal. In any event, he's happy to leave the challenge of developing HAL-like computers who recognize speech or express emotion to his colleagues in artificial intelligence: None of it, he says, has much to do with automated reasoning."

However, it seems to me that AI has its use in Automated Theorem Proving (which has been already shown by [2] and [3]). AI is not science fiction, it's real and it has its successes in scheduling, speech recognition, planning, etc. Why not let AI make its magic in theorem proving?

In this paper I will present a system which is correct as a logical based system with artificial intelligence algorithms to reason in specific domains.

2 Artificial Intelligence in Automated Reasoning

As seen before, two approaches can be followed for Automatic Theorem Proving, however it seems that the one based on logic reasoning is the most popular. This is not odd at all since most of the researchers in this field have a mathematics background which is heavily based on abstract reasoning.

Nowadays, most theorem provers seem be built on certain features which make them good and attractive for researching, however limits them to a certain extent. These limits will be briefed on the next subsections.

2.1 Generality

Most theorem provers publicize themselves[5] as being able to work on mathematics, hardware and electronic circuits, software verification, etc. These systems do not have the possibility to introduce restrictions of a specific domain for a greater efficiency and power.

Although one may want sometimes a general theorem prover that may not be the case. So sometimes, specific tools for specific tasks should be at hand. And in that case that tool will be able to provide the user a better service since it was implemented only for that specific job. For example, one can search for the satisfiability of a boolean formula in many ways or implementing a naive algorithm but that won't beat a SAT solver, which is a specific system for this specific job.

So, the best way to handle this generality problem is to have a set of specific tools that will handle specific inputs to the system. If the system handles 3 specific areas, the initial input parser to lead the input to the correct solver. Each of the solver may even reduce the problem to one of the other areas, leading then the new problem to another solver.

2.2 Abstraction

Power of abstraction is something which every mathematician must have to prove theorems, however, automated theorem provers seem to lack that very important feature. Without it we will always be reasoning on a very low-level language without being able to reach the interesting theorems which are located on a very high-level mathematics. Being general doesn't help an abstraction based theorem prover but like I said, a specific one might use those specific domain features to create different levels of abstraction.

Imagine the levels of logic, arithmetic and number theory. These may be viewed as 3 different degrees of abstraction where one may base arithmetic on logic and number theory in arithmetic. So, logic is the lowest level of abstraction and number theory is the highest level of abstraction. The interesting thing about abstraction is that after defining the arithmetic operations and concepts with logic and number theory operations and concepts with arithmetic, we rarely need to go below the current level of abstraction to prove something. The idea is that each level of abstraction defines a language and axioms where a theorem can be proved. Those axioms will be defined with the concepts, axioms or theorems of lower levels, in this way we create an acyclic directed graphs of levels of abstraction and their dependencies.

2.3 Repetition

Repetition is one of the main features of nowadays theorem provers maybe due to its logical approach. If we try to prove a theorem once and we are out of luck, we will still be out of luck if we try to prove it again and again on the same theorem prover until the end of time, or maybe until a new release is out which might include a new inference rule that matches that particular theorem.

However, recent research[2] already tries to learn from previous proof experience so that, new theorems similar to theorems already proved are proved in the same way. Even though there are very difficult problems with this I think that letting go some of the automation and ask the use in a compreensible language where to go past some point would be the way to go.

2.4 From Automation to Intelligence

Artificial Intelligence has greatly evolved since the 50's and new advanced algorithms in planning and searching have been developed. Although this changes the approach taken by most theorem provers, it can introduce a new level of successes in the area of Automatic Theorem Proving and the "Automatic" process can become "Intelligent".

3 Importance

The system is not only important from a research point of view. Its success or failure might show how the fields of AI and ATP intersect (some consider ATP to be a subfield of AI). If successful, its use might be of relevant for mathematicians in multiple areas for proving non-trivial theorems or just to find alternative proofs for theorems already proved and it might also be used for teaching purposes on theorem proving and mathematics. However, if not successful, the effort is not in vain, since the work on interfacing the several components of the system and the algorithms used in the development of an automated theorem prover using AI techniques will be interesting and original.

4 The System

The system will be developed as a plug-in of the Isabelle logical framework, much like `blast_tac` was developed[6]. It will be automatic (in that it will not interact with user), based on second order logic and it will use Isabelle to check the soundness of the proof. In this way, one will not have to worry about proving the soundness of the system, which would not be trivial on an artificial intelligence based system. The system will generate a proof of the current theorem and run it through Isabelle, only then if accepted by Isabelle, it'll be shown to the user so that incorrect proofs do not get through.

The system will remember previous successes on theorem proving so that doesn't have to repeat a proof it has already done. For this task it will use FDL (similar to NuPRL5), configured to have Isabelle as a refiner[7], or any similar database.

The system will be developed using SML, since it is the standard language to interface with Isabelle and NuPRL like systems.

This work will go through 3 phases:

1. Interfacing Isabelle with a mathematical database to store previous proofs and valuable information. This is essential if one want to have persistent objects like definitions and theorems. This way it will not be necessary to repeat the same proof all over again since it is already registered on the library.
2. Integration of the SML module with Isabelle. This was already accomplished for `blast_tac`. The integration has two aspects[6]:
   - translation of Isabelle rules to the system rules.
   - translation of the system proofs to Isabelle proofs.
3. Implementation of the system in SML.

The system will initially receive from Isabelle the theorem to prove and after checking it was not proved before using the theorem database it will prove the theorem and then send the completed prove to Isabelle which will run it and only show it to the user if it is correct, otherwise will fail. This assures the user that he will only be presented with correct proofs.

5 Conclusions

The system will try an approach most systems today do not follow. I will implement AI algorithms to prove theorems in a mathematics specific domain (initially Algebra and Real Analysis) and hopefully the field of AI will contribute much to Automated Theorem Proving.

Although the system is not complete (due to second order logic constraints) and its soundness is only assured by Isabelle, it will have a lot of other features. It will be able to store proved theorems, definitions and other mathematical knowledge, it will be able to create abstraction levels (with the help of the user) and to generate proof certificates explaining the user why the theorem is valid.

This is a work under heavy progress so every detailed information about it might be somewhat deprecated after some time. However, here I tried to show how I imagine the system in its final state and I am trying to implement it.

References

1. Davis, M.: The early history of automated deduction. In Robinson, A., ed.: Handbook of Automated Reasoning. The MIT Press (2001) 3–15
2. Denzinger, J., Fuchs, M., Goller, C., Schulz, S.: Learning from previous proof experience: a survey (1999)
3. Bundy, A.: The use of explicit plans to guide inductive proofs. In: CAD. (1988) 111–120

4. Obermiller, T.: Top of his game — biography of larry wos. `http://www-unix.mcs.anl.gov/~wos/uofc-bio.html` (1997)
5. McCune, W.: Otter 3.3 Reference Manual. (2003)
6. Paulson, L.C.: A generic tableau prover and its integration with Isabelle. Journal of Universal Computer Science **5** (1999)
7. Allen, S.F., Constable, R.L., Eaton, R., Kreitz, C., Lorigo, L.: The nuprl open logical environment. In: Conference on Automated Deduction. (2000) 170–176