Agent-Based Real-Time Pedagogy for Proof Construction

Paul Bello and Selmer Bringsjord

The Minds and Machines Laboratory
Department of Cognitive Science
Department of Computer Science
Rensselaer Polytechnic Institute
Troy NY, USA 12180
bellop@cs.rpi.edu, selmer@rpi.edu

There is a disturbing paradox at the heart of contemporary American education: As this education turns more and more "electronic," we are moving away from the one kind of learning that we know to be most effective, namely, one-on-one instruction. As the need for good teachers at the university level continues to grow, we see this paradox intensifying. And we see the problem manifesting itself in a particularly nasty way in curricula that predominantly focus on cultivating abstract reasoning ability in future scientists and engineers. The data tells us that as educators, we are not producing students able to successfully employ context-independent reasoning in technical domains. This is true despite the fact that there has been great progress made in developing educational technologies and aides for teaching formal, context-independent deductive reasoning; we refer here to an abundance of proof-construction environments. The fact is, teaching students to be good abstract reasoners requires the professor to have a one-on-one relationship with each student, with a keen eye on how each searches for a solution. The perfect automated logic instructor should be adaptable, and fully available to each student, at every time and every place. This is obviously not possible with human instruction, but our preliminary work suggests that our vision is capable of being realized in the digital domain: We are developing a suite of intelligent agents that bring the cutting edge in AI-based tutoring to the state-of-the-art in proof construction courseware. In addition, with agent-driven tutoring systems as a foundation, we aim to extend our agents so that they can be of assistance to logicians, mathematicians, and computer scientists in their research and development. Unfortunately, proof-construction environments in the educational realm, while presenting lucid proofs to the student, are based on weak theorem provers – provers that lack the sheer muscle to be of use to a professional scientist or engineer. We remedy the situation by using "industrial grade" theorem provers as the testbed for the development of our artificial assistants.

Introduction

A capacity for first-rate context-independent reasoning is a vital skill for success in today's high-tech society, which increasingly demands that people solve problems far removed from concrete contexts[19]. Despite the fact that this skill is supposedly taught from high school on through college, there is a remarkable dearth of good context-independent reasoners in our college classrooms. This is clearly demonstrable through a simple problem such as Wason's Selection Task[22] :

> Suppose that I have a pack of cards each of which has a letter written on one side and a number written on the other side. Suppose in addition that I claim the following rule is true:
>
> - If a card has a vowel on one side, then it has an even number on the other side.
>
> Imagine that I now show you four cards from the pack:
>
> 
>
>
> Which card or cards should you turn over in order to decide whether the rule is true or false?

Cheng and Holyoak conducted studies of this nature, with disparaging results that fully support our claim[8]. Only approximately 5% of the educated population solves this problem correctly, even after having an introductory course in logic. It seems safe to conclude that we're just not doing a good enough job teaching our students how to formally reason. Our preliminary work seems to suggest an antidote, because it has shown that an agent-based presentation of material achieved better results on a post-test among an experimental group of subjects than those who learned the material in standard lecture format[17]. Our motivation is to deliver a mature brand of this agent driven instruction to the masses, so that the absence of a human instructor who needs to attend to individual students has positive, rather than negative, effects.

Harnessing the Power of One-on-One Tutoring

One-on-one tutoring is remarkably effective: we have long known that there is strong evidence from a myriad of domains that tutored students consistently outperform those taught in classroom situations having standard student/teacher ratios. For example, in a meta-analysis conducted by Cohen, Kulik & Kulik of 65 evaluations of school tutoring programs, it was shown that the median tutored student performed at the 66th percentile of the untutored students[9]. Bloom discovered that the average one-on-one tutored student performed two standard deviations higher than students in normal elementary school classroom environments. Studies specifically concerning machine tutoring systems have long followed suit. Woolf reports a one-sigma improvement by students learning via tutoring systems[23]. Parallel results have been

obtained with respect to the Lisp and geometry tutoring systems developed by John Anderson and his team at Carnegie-Mellon University[1].

The Dilemma

The notion of trying to build an intelligent tutoring system for logic has a long and checkered history. Intelligent tutors developed for our domain of interest; logic, seem to be lacking in three fundamental areas: those areas being interactivity, powerful theorem proving engines, and the ability to let the student explore multiple lines of reasoning before interfering. Specifically, competency in logic requires innovation on the part of the student. Knowing what assumptions to make in a proof to achieve a certain end incurs not only the cognitive cost of knowing a set of rules to apply, but adds the effort of examining the proof from numerous different angles. This type of introspection is crucial to what is commonly referred to as "deep learning". We are working on a way to solve all three of these fundamental problems, and are packaging the solution in the form of an embodied intelligent agent. The realization of this solution has taken its preliminary shape in an agent-driven proof construction environment called *The Rensselaer Intelligent Prover* or RIP for short. Of course, a system such as RIP comes with a list of modules that are among the most difficult problems in computer science to implement. Among them are natural language processing, advanced prediction capabilities, and effective user interface design. Many of the best attempts at constructing such a system have a common ingredient for success missing from their software, namely the ability to *do* what a human professor does best. One of the primary advantages of the human instructor is the ability to provide natural language clarification of concepts, and the wealth of domain knowledge that is vital in the evaluation of a student's performance on a given exercise. Our best instructors have an uncanny knack for adapting to individualized need on a per student basis and keeping an eye on how the student progresses over time as concepts are reinforced through new material. A simple example would be a student who hasn't yet mastered *Modus Tollens*, which simply states the following:

$$\frac{P \rightarrow Q \text{ (if P, then Q)} \qquad \neg Q \text{ (it is not the case that Q is true)}}{\therefore \neg P \text{ (therefore it is not the case that P is true)}}$$

This is a vital rule in the simple derivation of P or not P ($P \vee \neg P$), which is vital to the solution of other more complicated derivations. A seasoned instructor would be able to look at one of these more complicated derivations, and immediately see that the student is incorrectly applying *Modus Tollens,* and provide a more fundamental example to reinforce the concept.

The ability to perform meta-reasoning is unavoidable within the framework of a computer instructor for any subject, but all the more vital for subject areas such as logic. These areas of study are plagued by a lack of a certain one-to-one mapping from problem statement to solution. The one-to-one mapping issue is easily illustrated by the fact that all first-order logic problems can be solved with indirect proof (proof by contradiction). If we have a proof that we want the user to construct, he is just as correct arriving at the answer via indirect proof as he is

using some other method of proof. In the ideal situation, we'd like to be able to let the student explore both lines of reasoning. Unfortunately, the ideal situation comes with a bit of overhead. We must have multiple valid solutions for a particular proof on hand, and have the appropriate automated advice hard-coded for these particular situations. Given a large corpus of proofs for our artificial instructor to choose from for example problems, we have an unmanageably large database. What's more, as the programmers, we have to either write out all of these solutions ourselves and enter them into the database, or go through the trouble of interviewing logicians. Even in the case of the latter, we run into the problem of each logician being predisposed to a certain way of analyzing a proof.

While it is our primary goal to develop an environment for automatic instruction in logic, we are proposing ancillary usage of our core technologies. These technologies lend themselves nicely to the domain of research and development for scientists, mathematicians, and engineers alike. Unfortunately, the computational muscle that is required to tackle the problems that these types of researchers face is not present in any of today's proof construction utilities. These scientists must rely on theorem proving utilities that usually lack elegant interfaces, and certainly lack the ability to make suggestions about the problem domain.

It has also been shown that tutors that are capable of non-trivial interactivity (a dialogue with the user for example) produce substantial learning gains among the vast majority of students who use them. The interactivity in question is all but missing from intelligent tutors in the domain of logic. A short exposé of previous efforts in this area will bring these shortcomings to light.

Previous Work

Since the advent of Herb Simon and Alan Newell's *Logic Theorist* at the original 1956 Dartmouth conference, researchers in AI have been pushing the boundaries of automated theorem proving – and things have been progressing nicely. ATP's (automatic theorem provers) have cracked mathematical theorems that would have taken mere mortals a lifetime to prove in a matter of days.

One of the more famous examples of one of these elusive theorems is the proof of the Robbins problem[11], which asked whether one set of rules is powerful enough to capture all of the laws of Boolean algebra.

One way to state the Robbins problem in mathematical terms is to ask if the equation "$\neg (\neg (P))=P$" can be derived from the following three equations:

1. $P \vee Q = Q \vee P$,

2. $(P \vee Q) \vee R = P \vee (Q \vee R)$,

3. $\neg (\neg (P \vee Q) \vee \neg (P \vee \neg (Q))) = P$.

This problem stumped our most brilliant mathematicians for sixty years, till the EQP (Equational Prover) system developed at Argonne National Labs by William McCune, cracked it in a mere eight days[15].

AI systems that once relied on proprietary (and oftentimes buggy) verification algorithms are now starting to harness the power of ATP, making rapid prototyping possible. But one of the things that theorem provers have not managed to do is teach a person how to create an abstract representation of a problem, and solve it through formal reasoning. (Perhaps one way to encapsulate the challenge is to say that we need theorem provers smart enough to both teach students the principles upon which they are based, *and* how to use these provers to solve problems.) However, some of these attempts laid the foundations for our pioneering work here at Rensselaer and at other institutions of higher education. Some of the standout efforts in the area of automated logic instruction are as follows

CMPT – The Carnegie Mellon Proof Tutor

The initiative at Carnegie Mellon University to develop an automatic tutoring system was the brainchild of Scheines and Sieg[20]. The system is built on top of the *Valid* theorem proving utility developed by Patrick Suppes at Stanford University[21]. CMPT (Carnegie Mellon Proof Tutor) was intended to replace a traditional first course in logic, and met with a surprising amount of success. These preliminary successes, as important as they were, don't add up to an immersive tutoring experience. CMPT can only be used with the propositional calculus (a subset of first order logic), and to date hasn't been upgraded to handle FOL. The interface to a past version of the tutor is shown in Figure 1 and illustrates some of the major shortcomings in interface design that run as a common thread through many automated tutors for logic. The proof is made much more difficult to read using the proof tree representation. There is a considerable lack of seamless interactivity. Looking at the figure, it stands to reason that the reader would need a basic background in logic just to understand the proof tree explanation of how to progress in the proof.

While the goal remains to teach a broad student base how to assemble proofs in first order logic, most of the brainpower of these programs is in the ATP behind the scenes. This does little to make the software easy to use, or the visual formalisms that it uses more palatable to the eye. In the case of CMPT, the preferred representation is a Fitch-style natural deduction proof (which is what we'd like to use as well), using windowing to encapsulate assumptions and their corresponding results in the proof. Unfortunately, even though CMPT can complete a proof from any given step in the reasoning process, this is hardly ground to deem it a tutor. Speaking as teachers, we find that during office hours when there is an opportunity to have a one-on-one session with a student, we resort to drawing out our lines of reasoning, and showing why certain assumptions work in certain situations. In the CMPT tutoring environment, all that the user sees is a goal tree, with the path of derivations from the premises to the conclusions. There doesn't seem to be any deep interactivity offered by this piece of software, such as rudimentary natural language capability or hypermedia presentations.

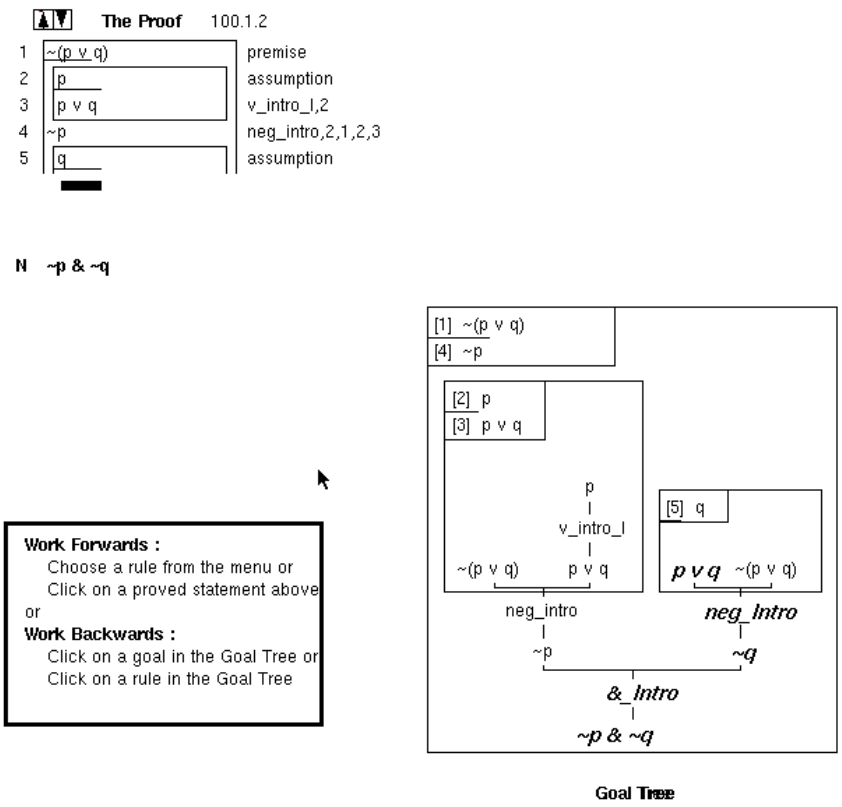

Figure 1: The Carnegie Mellon Proof Tutor

CSLI Initiatives

The industry standard in proof construction software is the excellent suite of software offered by the CSLI group at Stanford university, and pioneered by Barwise and Etchemendy (Hyperproof[4] and Tarski's World[3] for the Macintosh platform, and more recently the java based Fitch[5] and Tarski's World systems. While this software provides a very flexible and intuitive proof construction environment, it does not afford any assistance from intelligent software agents, or tutorials of any kind of the *in silica* variety. The CSLI software is accompanied by a textbook, and has been employed with a large degree of success in many colleges. Proofs are carried out in Hyperproof in the Fitch style of natural deduction, and in an easy-to-read fashion. Well-deserved attention was paid to developing a remarkably easy-to-use interface. Hyperproof provides support for developing visual proofs as well, which is arguably one of the most revolutionary and ground breaking features to ever appear in a piece of logic courseware. This leads to the exciting possibilities of constructing disproofs as well as students being able to tinker with situations, thus refining their ability to reason and anchoring the formalisms in a visual depiction (Shown in Figure 2.)

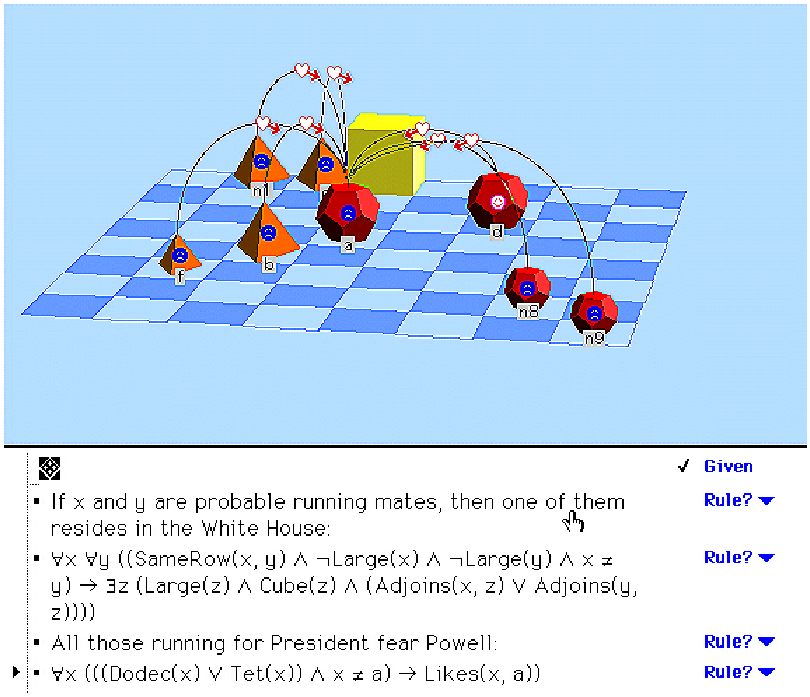

Figure 2: A Sample World in Hyperproof

A Comparison of ITS for Logic Instruction

There is much in the way of software systems that claim to be "intelligent tutoring systems" or something akin to the latter. To clear up the confusion, we've investigated many of the aforementioned systems, and have developed a comparison of ITS software systems (presented below in Figure 3).

System/Property	Strong Theorem Provers	Semantic Components	Multiple Types of Reasoning	Real-Time Advice	Offline Advice – Lecture
CMPT					
Fitch					✓
Hyperproof		✓			
WinKE					
RIP	✓	✓	✓	✓	✓
CHOGIC	✓	✓	✓	✓	✓
Existential-Graph Bvased Provers	✓	✓			
DPL-Based Systems	✓		✓		
Tarski's World		✓			

As can be seen in the figure, there are two main types of logic instruction software; that which already exists (Fitch, Hyperproof, WinKE*, CMPT, Tarksi's World, et al.), and that which is under development (most notably the Rensselaer Intelligent Prover and CHOGIC*). The systems under development are driven by artificial agents that are capable of giving advice, whether that advice be real-time, lecture-based, or offline (probably based on the student's historical performance on a certain type of question). Another common property of the systems under development is the tendency to use powerful theorem proving utilities as back-ends for the software. This provides a greater robustness to the overall software package. We have also included a column for "semantic component". This can include either visual reasoning using such systems as Hyperproof or Tarski's World, or it can be contextualized reasoning engines such as the one found in the CHOGIC system.

The Future is now: Agent Driven Logic Instruction

In our approach, central to the construction of an effective tutoring system is the notion of an intelligent agent. This agent is intended to behave as ways directly analogous to the human logic instructor. In particular, our focus is on building intelligent agents able to offer real-time advice on a one-on-one basis. As such, these agents are responsible for monitoring the progress of the user, and for providing advice and hints. We are also in the process of building agents that give traditional-style lectures, and review assignments submitted by students. (Recall the comparison we introduced above.) Since all of our work is based on the intelligent agent paradigm, and since, in particular, intelligent agents come in many varieties, a brief review of the paradigm is necessary. We begin with:

"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors. A human agent has eyes, ears, and other organs for sensors, and hands, legs, mouth, and other body parts for effectors. A robotic agent substitutes cameras and infrared range finders for sensors and various motors for effectors. A software agent has encoded bit strings as its percepts and actions." [18]

The traditional definition of an agent is so open-ended that almost anything with some input/output capabilities may be labeled an agent. It is therefore useful to examine some of the most common types of intelligent agents, enumerate the set of necessary tasks that a human instructor must be capable of to be a good teacher, and finally to match up these tasks with the capabilities of the aforementioned agents. Let's have a look at some different types of agents and what they can do for us.

Reflex Agents

In the preliminary stages of the project, our agent will manifest itself as a simple reflex agent with some enhancements that we will proceed to describe in detail. It is often the case that explicit lookup in a table is insufficient for making quick inferences. This is especially true in

* A proof tool based on the semantic tableaux developed by Ulle Endriss at King's College, UK[10]

* CHOGIC is a proposed system to teach different types of logic and reasoning using the rules, tactics, and strategies in chess[6]

our case, where variables can take on many different names, and the space for naming predicates is virtually unlimited. The objective is to reduce the search space to a manageable size while maintaining correctness and precision. This is accomplished through the use of a reflex agent. A reflex agent (Figure 3) is simply a collection of *condition-action* rules that have the form: if condition then action.

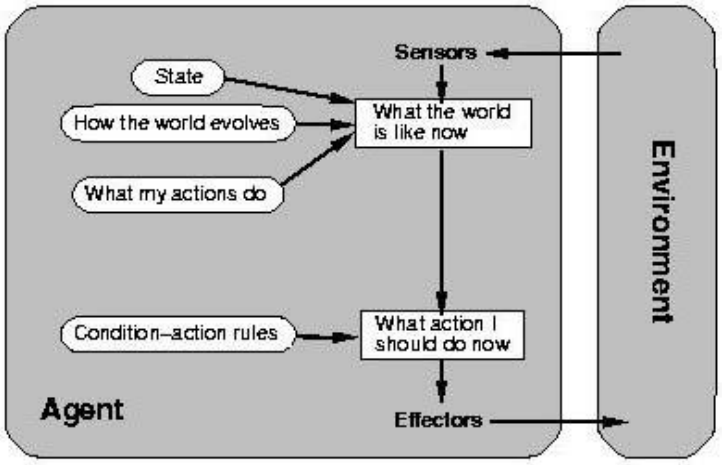

Figure 3: Reflex Agent with State

These rules govern the behavior of the system with respect to its percepts and constitute the agent's abstract view of the world. This architecture has the drawback of only being able to respond to singular situations and is unable to act with respect to context, or to a string of correlated events. For our purposes, we need an agent that is able to make a judgment based on the current information it has about what a user has done thus far in a problem situation. This "internal state" defines each step of a proof in the context of what has already been attempted. The basic operation of such an agent is shown in Figure 4. This will be our basic architecture for our first experiments.

```
function REFLEX-AGENT-WITH-STATE( percept) returns action
  static: state, a description of the current world state
          rules, a set of condition-action rules

  state ← UPDATE-STATE(state, percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  state ← UPDATE-STATE(state, action)
  return action
```

Figure 4: Algorithm for Reflex Agent with State

Goal-Based Agents

At the heart of all tutoring systems lies the *a priori* knowledge of expert(s) in the chosen domain. A tutoring system for logic will often have explicitly coded solutions for each exercise that a

student is presented with. We feel that this is a limiting factor for how useful the system we envision can be. An agent should be able to take the current state of the proof, and generate a sequence of steps to eventually satisfy all of the goals. The ability to perform such a complex chain of inferences is the hallmark of a goal-based agent. The goal-based agent architecture that we endeavor to implement subsumes both traditional planning agents and utility-based agents. While our agent needs to be able to map out a sequence of steps to get from where the agent decides to intervene to the goal state(s), it must also be able to select advice that can clearly be comprehended and implemented in the context of the student's progress through the problem. It is therefore necessary to have an agent that can plan, but also has a sense of what action will result in the most positive outcome for the student.

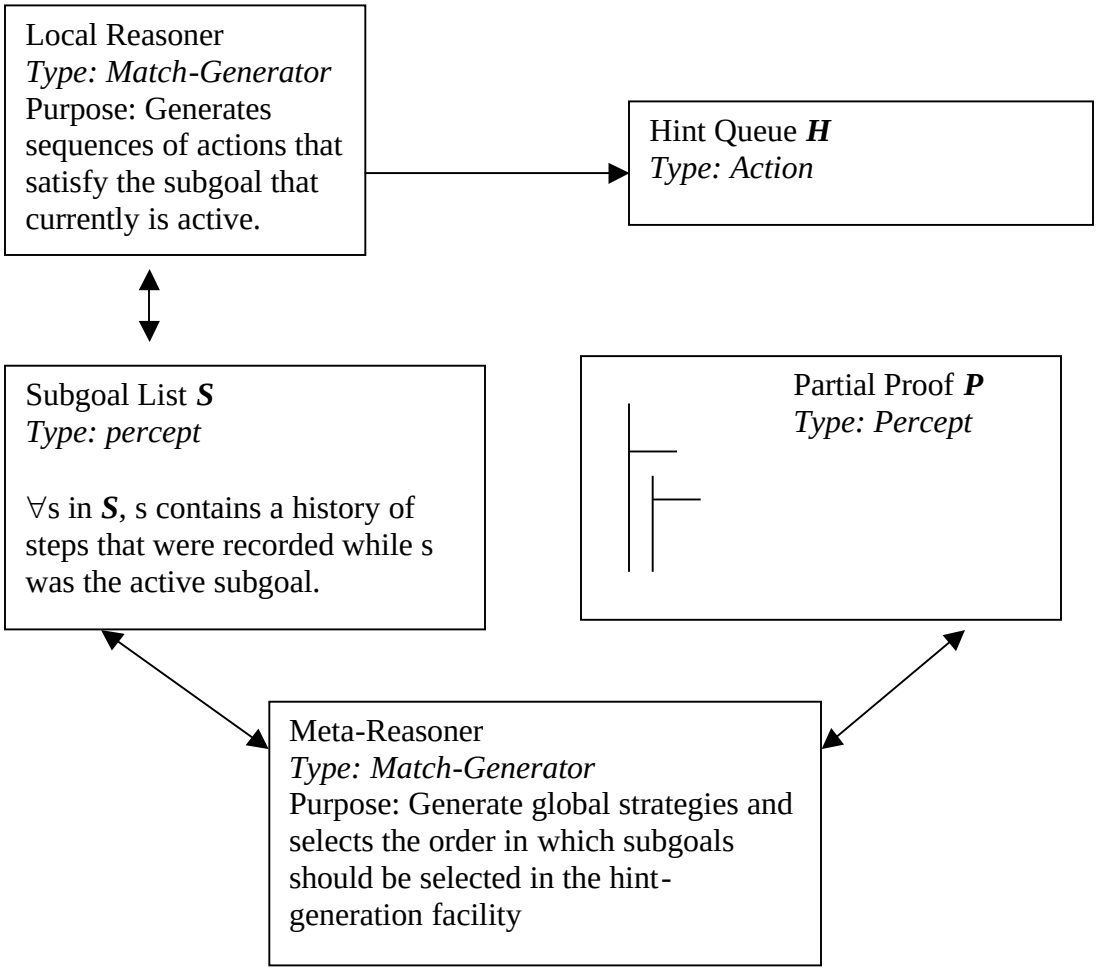

Figure 5: Our integrated agent architecture

Figure 5 illustrates our augmented reflex-agent architecture. As you can see, our agent relies on a divide and conquer approach to giving hints. If asked for a general hint as to how to complete the proof, the meta-reasoning facility will examine a snapshot of the current state of the proof, and produce an ordering of subgoals that are necessary to complete the proof in a predefined

manner, taking into account the general teaching strategy (proof by contradiction, conditional proof, et al). If we refer back to Figure 4 for a moment, we sketch a brief algorithm that our agent uses for RULE-MATCH and RULE-ACTION below:

Algorithm: Hint-Generation
Parameters: Subgoals ***S***, Partial Proof ***P***, Active subgoal ***s***

```
Variable_1 = Is_Current_Subgoal_Satisfied?
        If(Variable_1 = TRUE)
        {
                Hint Queue H ← "Active subgoal is already satisfied. Please go on."
        }

        Else
        {
                Variable_2 = Solveable_From_Available_Info?
                        If(Variable_2 = TRUE)
                        {
                                List_of_Steps ← Local_Reasoner(s, P)
                                Hint Queue H ← List_of_Steps
                                Presentation ← Query_Advice_Database(Hint Queue H)
                                Present_Hint(Presentation)
                        }

                        Else
                        {
                                Meta_Reasoner(S – s)
                                Hint-Generation(S, P, s)
                        }
        }
```

The Microsoft Agent

Having an agent with all of these capabilities, we needed to find a robust embodiment to act as our vehicle for disseminating information. The vehicle of choice in this case turned out to be the Microsoft Agent. Microsoft Agent is a programmable desktop caricature that is capable of speaking, moving, gesturing, being spoken to and responding. This relatively rich set of behaviors is a more than suitable way to embody the agent architecture we've mentioned previously. Our agent of choice is Robby the robot, courtesy of Microsoft Research[16]. Robby is shown in Figure 6. The agent is run by Visual Basic code that is embedded in the web pages that RIP displays in its browser window.

Figure 6: Robby, our Hardworking Assistant

We are using the full suite of Microsoft Agent functionality in the algorithm design. Robby will respond both to voice commands and to a help button on the application. The infrastructure of the agent is based on the algorithm we've sketched out above, and provides the user with one or more steps in the progression of the proof, depending on how much help is requested.

The Rensselaer Intelligent Prover

As the reader has seen, there are plenty of proof construction tools out on the market already, but for the purposes of proof-of-concept, the Minds and Machines Laboratory has undertaken the development of a testbed for our AI. The result of this work is the Rensselaer Intelligent Prover (or RIP for short). RIP arose from the need to have an interface capable of supporting the type of rich multimedia experience that we've advocated in this document plus the power of high-grade theorem proving wrapped in an intuitive interface. While RIP is by no means robust enough to be deployed in the classroom for full-time use, it serves its purpose as a testing platform for our collection of agents and the gathering of data.

Our First Experiment

For logic, we experimented with the role of an artificial agent in teaching students who do not have a physically present instructor. The agent we designed (pictured in Figure 7) was used to teach a specific topic, proof by contradiction (or *reductio ad absurdum*), to students from the RPI *Introduction to Logic* class. After dividing volunteers into two groups matched for ability in logic by a pre-test (the 1998 version, differing only in date, is available at http://www.rpi.edu/~faheyj2/SB/INTLOG/pre-test.f98.pdf), we offered the students in the experimental group an hour-long course in the use of the software, without presenting to them any content. The purpose of this training was to ensure that the hour they would have later for instruction would not be compromised by a lack of familiarity with the mechanics of using the interface, which was in some ways unlike the other programs used in the course. We then gave each group simultaneous instruction--the experimental group viewed the interactive software, the control received instruction as normal from their professor. Following the instruction, we gave each participant a post-test consisting of a logic proof that required understanding of the concept discussed in the lessons (again, proof by contradiction). Three of six students in the experimental group received full credit; only one of seven did so from the control, with another earning partial

credit. Relatively low attendance rates complicated the statistical interpretation somewhat, but the difference was reported at a significance of .092, indicating that it was quite unlikely that in-person instruction was better than instruction by the artificial intelligent agent.

Though we did expect that students would not do significantly better when taught by a physically present human instructor, it came as a welcome surprise that even this early version of the software was able to substantially outperform an experienced full professor with over ten years of experience teaching this subject (achieving twice the success rate).

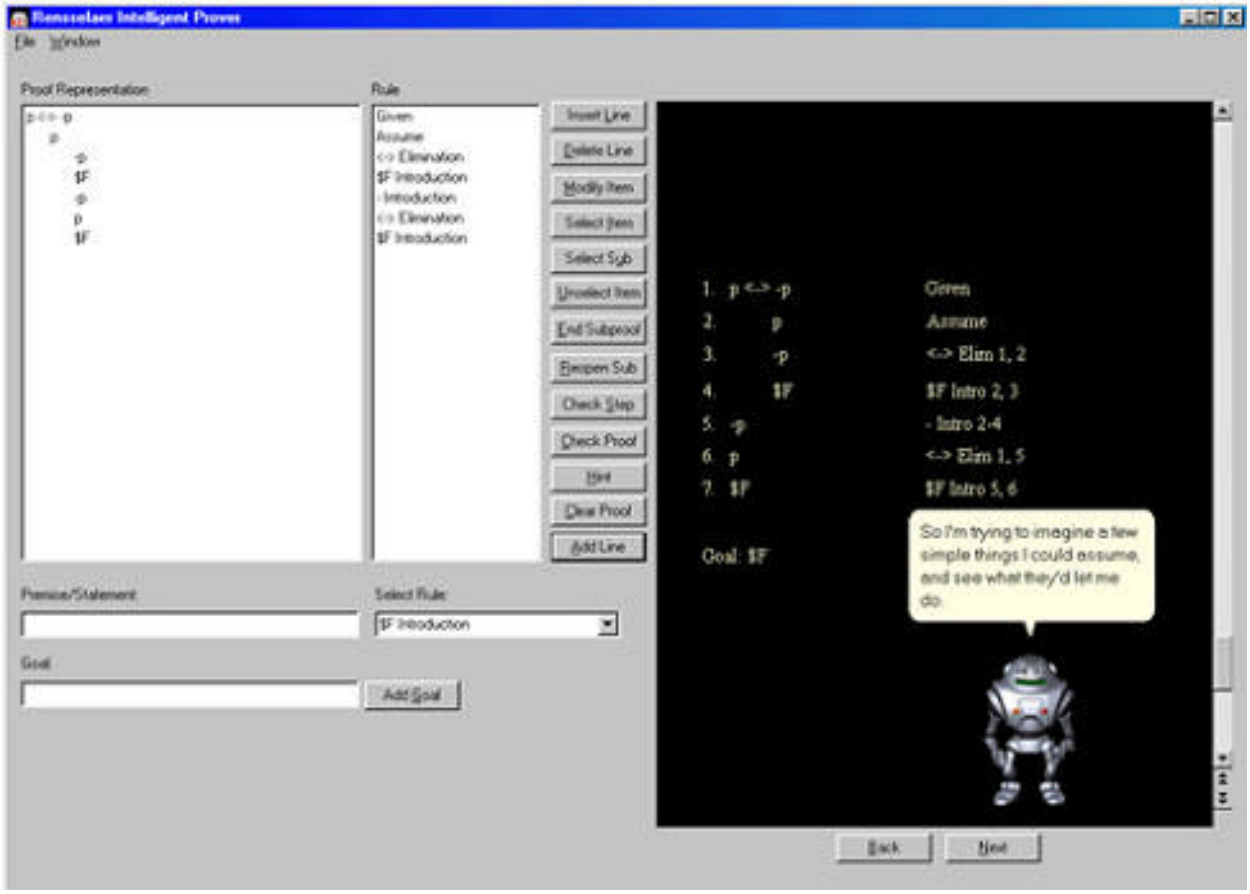

Figure 7: The Rensselaer Intelligent Prover in Action

Powerful Theorem Proving

So far, we've concentrated on the problem of minimal interactivity in ITS's for logic, but there is still a glaring fault that we've yet to address, namely the lack of powerful theorem provers as back-end processors for the interface. The problem presents itself rather forcefully, especially

when untrained logic students (such as those used for our experiments) are routinely able to "break" proof construction tools such as Hyperproof on a routine basis using only the propositional calculus. These malfunctions occur with an even more frightening frequency when quantification is introduced. The Robbins problem, which was previously mentioned, is a perfect example of a problem that requires more computational muscle than a system such as Hyperproof is able to muster. This doesn't bode well for the extension of these systems to robust, bulletproof tutoring facilities. As it turns out, the authors have some hands-on know-how that circumvents this difficulty. We have been in the business of interfacing some of the best theorem provers that are available with proprietary applications since 1997 using William McCune's OTTER theorem prover[14].

OTTER

OTTER (Organized Techniques for Theorem-proving and Effective Research) was developed by William McCune in 1994 at the Argonne National Laboratories for use as a high-powered tool for proving first-order theorems with equality. We have employed OTTER in a number of different applications here at RPI, including *The Rensselaer Intelligent Prover*. OTTER has also been successfully used in the teaching of Logic Programming Courses[7] and various other AI-related courses. OTTER is a resolution-based theorem prover, and subsequently produces output that isn't what we would consider to be aesthetically pleasing. To better illustrate our point, let's take a look at some sample OTTER output:

```
---------- PROOF ----------
22 [] -At(STENCH,x,y)| -At(OK,INC(x),y)|
-At(OK,x,INC(y))| -At(OK,DEC(x),y)| -At(OK,x,DEC(y)).
30 [] At(OK,3,1).
35 [] At(OK,4,2).
36 [] At(STENCH,4,1).
40 [] INC(1)=2.
46 [] INC(4)=5.
59,58 [] DEC(4)=3.
65,64 [] DEC(1)=0.
79 [] At(OK,x,0).
82 [] At(OK,5,1).
87 [para_from,40.1.1,22.3.3,demod,65,unit_del,79]
-At(STENCH,x,1)| -At(OK,INC(x),1)| -At(OK,x,2)| -At(OK,DEC(x),1).
304 [para_into,87.2.2,46.1.1,demod,59,unit_del,36,82,35,30] $F.
---------- end of proof ----------
```

As you can see, proofs that are generated via resolution have been modified from their original form. In light of these modifications, these generated proofs become much more difficult to use in the context of an intelligent tutoring system to teach deductive reasoning. On the contrary, the type of output that OTTER produces does not rule out its use for our purposes. OTTER has been deployed as a back end in our Rensselaer Intelligent Prover to verify the syntactical correctness of user input (i.e. for the validation of steps in the proof), and to determine if some statement follows from the justification statements that are selected at that step of the proof.

Evolutionary Learning of Natural Deduction Proofs

One of the major obstacles to overcome in the development of a final release will be to give our agent the ability to generate multiple internal models of a proof scenario. In most tutoring systems, there is a definite set of steps to take from start to finish in a question. Sometimes this is just a simple question/answer scenario, and sometimes it is a set of definite steps that comprise the only correct sequence to get from the beginning to the end of a problem. In our scenario, a student may try many different approaches on the landscape of valid solutions. Often in these scenarios, it is perfectly acceptable for the agent to intervene when a user deviates from the correct sequence. This is obviously not the case given our domain of proof construction. One of us (Bello) is looking into the correlation between proof and program, so that it may be exploited to "evolve" a population of proofs based on the genetic programming methodology developed by John Koza[13]. The basic evolutionary algorithm is given below:

Start

Time unit $t \rightarrow 0$
initialize population P(t) randomly from our set of operators and variables
evaluate each member of P(t) by how well it satisfies our constraints

while stopping condition is not true {
$t \leftarrow t + 1$
select P(t) from P(t - 1)
transform selected individuals in P(t)
evaluate P(t) }

End

These evolved proofs are generated with a stochastic algorithm, and hence may produce proofs that were previously undiscovered by logicians and automatic theorem provers. Evolved proof structures will be automatically discovered by the system, and may be used by mathematicians to solve previously unsolved dilemmas. Having proofs that are evolved on the fly also eliminates the need for having a large corpus of pre-solved proofs on hand, and rids us of the time (and money) expenditures that we would incur in the hiring of experts to populate our database with solutions.

REFERENCES

[1] *Anderson, J.R., Corbett, A., Koedinger, K.R., & Pelletier, R. (1995),* 'Cognitive Tutors: Lessons Learned', Journal of the Learning Sciences, *Vol 4, 167-207.*

[2] *Arkoudas, K.(2000),* "Denotational Proof Languages", *MIT, Cambridge, MA.*

[3] *Barwise, J. & Etchemendy, J. (1991),* Tarski's World, *CSLI, Stanford, CA.*

[4] *Barwise, J. & Etchemendy, J. (1994),* Hyperproof, *CSLI, Stanford, CA.*

[5] *Barwise, J. & Etchemendy, J. (1999),* Language, Proof and Logic, *Seven Bridges, New York, NY.*

[6] *Bringsjord S., Yang, Y. (2001),* "CHOGIC: A Chess-Based Approach to Teaching Context Independent Reasoning".

[7] *Bringsjord, S. (1998),* Introduction to Logic Programming Course Website, *http://www.rpi.edu/~brings/ilogprog.html*

[8] *Cheng, P. W. , Holyoak, K. J., Nisbett, R. E. & Oliver, R. M. (1986),* 'Pragmatic versus Syntactic Aproaches to Training Deductive Reasoning', Cognitive Psychology, *Vol 18, 293-328.*

[9] *Cohen J., Kulik J., & Kulik C. (1982),* 'Educational Outcomes of Tutoring', American Educational Research Journal, *Vol 19, 237-248.*

[10] *Endriss, U. (2000),* "The Interactive Learning Environment WinKE for Teaching Deductive Reasoning", Proceedings of the First International Congress on Tools for Teaching Logic, *Salamanca, Spain.*

[11] *Huntington, E. V. (1933),* 'New Sets of Independent Postulates for the Algebra of Logic.', Trans. Amer. Math. Soc. *35, 274-304.*

[12] *Huntington, E. V. (1933)* , 'Boolean Algebras: A Correction.', Trans. Amer. Math. Soc. *35, 557-558.*

[13] *Koza, J. (1990*), 'Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs.', Stanford Computer Science Dept., *Stanford, CA.*

[14] *McCune, W. (1994),* OTTER 3.0 Reference Manual, Technical Report, Argonne National Laboratory.

[15] *McCune W. (1997),* EQP 0.9 Users Guide, Technical Report, Argonne National Laboratory.

[16] *Microsoft. (1999),* Microsoft Agent: Software Development Kit, *Redmond, WA.*

[17] *Rinella, K., Bringsjord, S. and Yang, Y. (2001).* 'Efficacious Logic Instruction: People are not Irremediably Poor Deductive Reasoners', Proceedings of the Twenty-Third Annual Conference of the Cognitive Science Society, *851-856, Lawrence Erlbaum Associates, Mahwah, NJ.*

[18] *Russell, S. & Norvig, P. (1994).* 'Artificial Intelligence: A Modern Approach', *Prentice-Hall, Saddle River, NJ.*

[19] *Stanovich, K.E. & West, R.F. (2000)* 'Individual Differences in Reasoning: Implications for the Rationality Debate,' Behavioral and Brain Sciences, *Vol 23, 645-665.*

[20] *Scheines, R. & Seig, W. (1990)* , 'Computer Environments for Proof Construction', Interactive Learning Environments, *Vol 4, 159-169.*

[21] *Suppes, P. (1981*), 'University-Level Computer-Assisted Instruction at Stanford 1963-1980', *IMSSS, Stanford, CA.*

[22] *Wason, P. (1966),* 'Reasoning', New Horizons in Psychology, *Penguin, Hammondsworth, UK.*

[23] *Woolf, B. (1988),* '20 Years in the Trenches: What Have We Learned?', Proceedings of ITS 88, *Montreal, 33-39.*

PAUL BELLO is a Ph.D. student in the Department of Computer Science at Rensselaer Polytechnic Institute (RPI) in Troy, New York. He specializes in both the logical and empirical foundations of AI, blending elements from both the logic-based symbolic approach and various machine learning paradigms.

SELMER BRINGSJORD specializes in the logico-mathematical and philosophical foundations of Artificial Intelligence and Cognitive Science. He currently is a professor in the Department of Cognitive Science, where he teaches artificial intelligence, logic, theorem proving, and philosophy of AI.