AUTOMATIC GENERATION OF INTELLIGENT TUTORING CAPABILITIES VIA EDUCATIONAL DATA MINING

by

John Carroll Stamper

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Information Technology

Charlotte

2010

ABSTRACT

JOHN CARROLL STAMPER. Automatic generation of intelligent tutoring capabilities via educational data mining. (Under the direction of DR. TIFFANY BARNES)

Intelligent Tutoring Systems (ITSs) that adapt to an individual student's needs have shown significant improvement in achievement over non-adaptive instruction (Murray 1999). This improvement occurs due to the individualized instruction and feedback that an ITS provides. In order to achieve the benefits that ITSs provide, we must find a way to simplify their creation. Therefore, we have created methods that can use data to automatically generate hints to adapt computer-aided instruction to help individual students. Our MDP method uses data from past student attempts on given problem to generate a graph of likely paths students take to solve a problem. These graphs can be used by educators to clearly understand how students are solving the problem or to provide hints for new students working the problem by pointing them down a successful path to solve the problem. We introduce the Hint Factory which is an implementation of the MDP method in an actual tutor used to solve logic proofs. We show that the Hint Factory can successfully help students solve more problems and show that students with access to hints are more likely to attempt harder problems than those without hints. In addition, we have enhanced the MDP method by creating a "utility" function that allows MDPs to be created when the problem solution may not be labeled. We show that this utility function performs as well as the traditional MDP method for our logic problems. We also created a Bayesian Knowledge Base to combine the information from multiple MDPs into a single corpus that will allow the Hint Factory to provide hints on new problems where no student data exists. Finally, we applied the MDP method to create models for other domains, including Stoichiometry and Algebra. This work shows that it is possible to use data to create ITS capabilities, primarily hint generation, automatically in ways that can help students solve more and more difficult problems, and builds a foundation for effective visualization and exploration of student work for both teachers and researchers.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Motivation

Our main goal is to improve the usefulness of computers for learning. In college and K-12 education, computer aided instruction (CAI) has become a mainstay in education today. In addition, intelligent tutoring systems (ITSs) that adapt to an individual student's needs have been shown to provide significant learning gains of one standard deviation over traditional computer based training (Conati et al. 2002) (Heffernan & Koedinger 2002) (Murray 1999). However, the adoption of ITSs in classrooms has been slow because these adaptive systems are extremely difficult to create. Studies have shown it takes between 100-1000 work hours to create 1 hour of content for an ITS (Murray 1999), and most of this time is spent on creating domain-specific cognitive models.   Our work allows developers to add some adaptive capabilities to CAI with minimal work, by utilizing past student CAI data to automatically generate intelligent tutoring capabilities. Our methods are domain independent, and can be readily applied in procedural domains. We demonstrate the extensive use of our methods in the domain of logic, and also show potential applications in Algebra and Stoichiometry.   We believe that our data-driven methods may also be useful in ill-defined domains where it is difficult to automatically determine if a student solution is correct or not.

1.2 Research Questions

In this section we present our research hypotheses, that all center around our primary hypothesis, given in Figure 1.1. Our central method, the MDP method for hint generation (fully described in Chapter 3), builds a "map" of steps in past student solutions and assigns a value to each step that determines how "good" the step is in terms of solving a problem. This map is used to guide new students by giving them

Primary Hypothesis: We can automatically generate ITS capabilities using educational data mining methods on past student data.

Figure 1.1: Statement of the Primary Hypothesis

hints on which step to take. Visualizing these maps, as shown in Chapter 4, can help educators better understand student learning and problem-solving. Our hypotheses for visualization are given in Figure 1.2.

Hypothesis 1: The generated MDPs can be useful to educators to identify student strategies for solving a problem.

Hypothesis 2: Educators will be able to better identify trouble spots using the MDPs.

Figure 1.2: Hypotheses for Visualization Research (Chapter 4)

As we show in Chapter 4, MDP visualizations help show the ways students solve individual problems. In Chapter 5, we hypothesize, as shown in Figure 1.3, that the MDP provides enough coverage of student work to allow hint generation for over 50% of student problem-solving steps in historical data sets. To generate a hint, we trace a student’s steps in the MDP where a past student worked the problem exactly the same way. Then, we look ahead in the MDP to see which future steps were most likely to lead to the solution, and use these to generate context-specific hints. In Chapter 5, we show that there is sufficient coverage to provide hints to a majority of students as an important step in verifying the MDP method as a potential source for generating intelligent tutoring capabilities. We also determine baseline measures for how much data is needed to attain different levels of coverage (e.g. percent of the time we predict hints will be available).

Hypothesis: We will be able to generate hints, from one semester of data, which would have been available 50% of the time if past students could have had hints on every problem step they took.

Figure 1.3: Hypothesis of Validation Research (Chapter 5)

We test the hint generator with students in real classes in Chapters 6 and 7. Our first experiment in Chapter 6 is a pilot study to investigate how students use hints, and shows that hints help students solve more problems than past classes without hints. In Chapter 7, a second experiment tests the hypotheses shown in Figure 1.4. The results of the experiment show that the hints not only help students solve the problems, but that students who have access to the hints in the first level tutor problems are more likely to attempt and complete the harder third level problems.

The remaining research in Chapters 8 through 11 describes work to enhance the MDP method to provide hints when full information about the problems is not known. In Chapter 8, we devise and test a method to generate hints when a problem solution is not known. We show that the utility work compares positively to our traditional method. The hypothesis is shown in Figure 1.5.

Hypothesis 1: System generated hints will help in overall learning of the material.

Hypothesis 2: Hints will improve students' ability to solve the proof problems.

Figure 1.4: Hypothesis of Experiments (Chapter 7)

Hypothesis: We will be able to generate hints, without knowing if students correctly solved the problem with accuracy that matches our traditional MDP method.

Figure 1.5: Hypothesis of Utility Research (Chapter 8)

In Chapter 9, we show that our ability to give hints can be quickly enhanced by having experts complete a small number of problem attempts to provide an initial seeding to the MDP. In fact, we were able to prove our hypothesis, shown in Figure 1.6, and achieve over 50% coverage of student states in the set with just three to four example solutions.

Hypothesis: : Experts will be able to get significant coverage (over 50%) for hints with just a small amount of time devoted to solving a problem by working examples.

Figure 1.6: Hypothesis of Seeding Experiment (Chapter 9)

In Chapter 10, we present a method to generalize the domain knowledge from individual MDPs to help provide hints for problems where we have no data. This BKB method breaks down several MDPs into smaller model components to create a Bayesian Knowledge Base (BKB). This BKB can then be used to provide hints for new problems for which no student data exists. In Chapter 11, we show how our approach can be applied to other domains. Specifically, we generate MDPs for Algebra and Stoichiometry tutors. Finally, in Chapter 12, we discuss the main contributions of this work in terms of the Cascading Hint Factory which is a theoretical framework for implementing our methods of automatic hint generation. We also describe the impact of this work by exploring research that has referenced this work, and chart our future directions.

CHAPTER 2: RELATED WORK

Our goals to provide an automated, data driven approach to building intelligent tutors are necessary in order to make the effectiveness of ITSs widespread. In order to understand where we are headed we present related work that deals with how ITSs are currently implemented. First, we discuss the ideas of learning that have been the foundations of ITS development. Then we explore the state of the art in ITSs including the cognitive tutors and constraint based tutors. Several existing attempts at logic tutors are explored since most of our work is done in this domain. Finally, we look at some of the tutors that have relied on machine learning techniques that also use data to build student models.

2.1 Learning

Within the field of cognitive science, which seeks to understand learning, much work has focused on how best to optimize knowledge acquisition. Work has been done on how to encode information in the computer in the way that the brain stores knowledge, like the declarative and procedural knowledge representations of Adaptive Character of Thought (ACT Theory (Anderson et al 1984). In the field of intelligent tutoring several works on learning have made an enormous impact. These include Bloom's work on tutoring (1984), Vygotsky's zone of proximal development (1984), and ACT-R theory (Anderson et al 1995), and we will discuss these here.

Bloom’s work focused on traditional classroom instruction compared to one-on-one tutoring (Bloom 1984). In this work he showed that one-on-one tutoring could increase student achievement by two standard deviations. This means that if a traditional lecture format class has a normal distribution of student achievement scores with the mean at the 50% percentile (of the curve not the class), one-on-one tutoring would have a mean two standard deviations better or close to the 98% percentile compared to the traditional class as seen in figure 2.1.

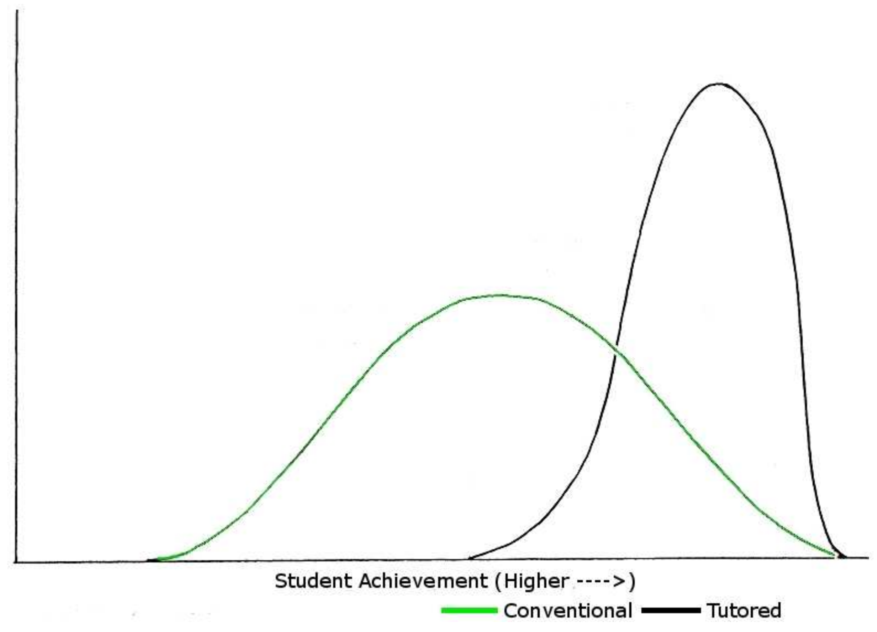


Figure 2.1: Difference in normal learning curve distributions of tutored vs. traditional classroom instruction. Modified from Bloom 1984.

Bloom suggested that this work clearly showed that traditional classroom teaching was not reaching all students and new methods such as mastery learning and tutoring were needed. Although this work did not focus on computer aided instruction or intelligent tutors the ITS field has used this work as a benchmark, trying to achieve results as good as a human tutor. Current ITS research has shown that intelligent tutors can achieve learning gains of better than one standard deviation (Koedinger et al. 1997).

Vygotsky's zone of proximal development (ZPD) states that students have the greatest chance at succeeding in material that is close to their current level (Vygotsky 1984). This concept is used to argue against standardized tests to measure intelligence, because of the "one test fits all" approach that is used. In teaching, proponents of ZPD believe that scaffolding can help students get closer to their individual zone. Scaffolding provides a learner with some framework of help, such as a partially worked problem or a worked example, which is similar to the construction use of the term where scaffolding means building a framework to support a structure. For example, for a math problem to find the area of a house composed of a triangle on a square, a scaffolding step would include the addition of a table to accept the intermediate areas of the triangle and the square. Scaffolding has been studied as a useful feature built into intelligent tutors (Rienburg and VanLehn 2006).

Since the early 1990's, some of the most effective ITSs have used a cognitive model based on Adaptive Character of Thought – Rational (ACT-R) theory (Anderson et al 1995), including the Carnegie Learning algebra tutor, (www.carnegielearning.com), the LISP tutor (Anderson and Skwarecki 1986), and the Ms. Lindquist algebra tutor (Heffernan and Koedinger 2002) to name a few. ACT-R theory makes a distinction between declarative and procedural types of knowledge. Declarative knowledge is composed of specific facts (such as 1+1=2) while procedural knowledge is composed of a list of steps required to accomplish a task. In designing an ITS, the use of declarative knowledge is simple to implement. Facts can easily be represented and programmed.

Procedural knowledge is much more complex because the same task can often be accomplished in many different ways. While people can generally state the declarative knowledge they have, they find it much more difficult to express procedural knowledge. ACT-R uses production rules to describe procedural knowledge. Production rules are statements that describe actions which should be taken if conditions are met (referred to as a condition-action pair). An example production rule is shown here:

```
If the goal is
        to classify a shape
        and the shape has three sides
Then
        classify the shape as a triangle
```

This simple example is a production rule that describes a problem where the student is classifying shapes.

Model Tracing tracks a student's progress through a particular problem. Its main use in an ITS is to keep a student on a successful path through the problem by giving them feedback to return to a good path if they proceed down an unsuccessful path or provide help if the student is stuck. With tutors that use the ACT-R framework, production rules are used to model student problem solutions or paths. These production rules can be "good" rules that are correctly applied, or "bad" rules, which generally signify a student misconception. The development of these production rules requires time and expert knowledge to create. Below are examples of a good and bad production rules (Mitrovic et al. 2003):

Good production rule:

IF goal is to find an angle in an isosceles triangle ABC and AC = AB and angle A is known
THEN set the value of angle B to A.

Bad production rule:

IF goal is to find an angle in an isosceles triangle ABC
and angle A and C are at the bottom of the triangle and angle A is known
THEN set the value of angle C to A.

2.2 Intelligent Tutoring Systems

The multidisciplinary field of intelligent tutoring systems has grown primarily out of the fields of Computer Science, Psychology, and Education and is at the convergence of computer aided instruction, human computer interaction, and learning theory (Wolfe 2009). Very early in the development of computers, people saw the potential for their use in teaching. Some of the first work to appear in the CAI field included the PLATO system developed at the University of Illinois in 1956 (Molnar 1990), and the drill and practice system developed at Stanford University starting in 1968 (Suppes 1981). Some of the first CAI to incorporate artificial intelligence were those based on expert systems such as MYCIN (Buchanan and Shortliffe 1984) and GUIDON (Clancey 1979) which both dealt with teaching medical cases. Anderson followed these systems in the early 1980's with his work on the LISP tutor (Anderson and Skwarecki 1986) which laid the framework of ACT-Star (Anderson 1983) which eventually became ACT-R (Anderson and Corbett 1995). These works became known as intelligent tutors and are now primarily referred to as cognitive tutors.

Intelligent Tutoring Systems are computer aided instruction systems that adapt to an individual student. Van Lehn described two loops that CAI can implement in order to be considered an ITS, the outer and inner loops (Van Lehn 2006). The outer loop refers to overall problem selection, ie. determining which problem the student should see next to optimize learning. The inner loop refers to helping the student solve individual problems.

According to Van Lehn, to be considered an intelligent tutor a system must, at a minimum, provide the inner loop. The adaptive features of an ITS during problem-solving (inner loop) consist primarily of help and feedback. Other adaptive features that could help in the inner loop of an ITS include reflection, debriefing, gaming detection, and tracking affect. Reflection is a step that asks the student to stop and justify the step that was just made. Using reflection in one on one human tutoring shows significant learning gains even when the tutor is not a subject expert (Graesser et al 1995). Several ITSs, such as AutoTutor (Craig et al 2007) incorporate reflection into tutoring process. Debriefing is a form of feedback given after a student has completed an entire problem or set of problem, which strives to allow the student to take away points to apply to a similar problem in the future (Bass 1998). Gaming is behavior where students try to out-smart the system and is often a sign of boredom. Gaming detectors have been built into several ITSs, and have been shown to effectively predict gaming and keep students on task (Baker 2007). Tracking affect is similar to gaming detection, but even more ambitious. The affect tracking tutors try to gauge how a student is feeling, and provide motivators to keep a student on task (D'Mello et al 2008).

Marking student work as “correct” or “wrong”, as is done in most CAI, is a simple form of feedback. Although help can be considered a type of feedback, giving students help is a much more complex problem. Studies of help seeking behiaviors in the classroom have been done (Karabenick, 1998), where the habits of help seeking students were studied, and strategies to improve help seeking were implemented. These studies showed that help seeking led to better student outcomes and that offering help to students on demand would increase the overall usage of help by students. Additional studies on help and feedback have shown that in order to positively affect learning outcomes the help and feedback must be directly relevant to the current situation that the student is facing (Shute 2008). The hints that our methods provide do provide context specific help that is relevant to the specific step and problem that a student is working on.

Help seeking behaviors have also been studied in the context of intelligent tutors with a number of tutors including the LISP tutor (Anderson et al 1989), and this research showed that tutors containing intelligent help messages improved the time students were able to complete the work by 30% with no loss in accuracy. Additional work on help and feedback with a cognitive tutor to teach Geometry showed that immediate goal directed help and feedback helped more than higher level conceptual feedback when learning gains were measured (McKendree1990). More recently, developers have sought to build more effective tutors by increasing good help seeking behaviors in students (Aleven and Koedinger 2000). This work showed that although students who asked for help had higher learning gain on average, there were behaviors that limited the usefulness of the help messages. These behaviors included, help abuse, help avoidance, and trying help too fast. The results of their initial model suggest that 72% of help seeking behavior is unproductive (Aleven et al 2004), and these bad help seeking behaviors could be mitigated, although their model has not yet been implemented into a live tutor in a classroom setting. Based on the hint and help research our implemented method provides the right type of help at the right time when a student needs it.

In order to give appropriate help when a student requests it, the software must have an adequate model of the problem as well as information on what the student knows. To give help, an ITS will generally include a domain model for solving problems and a student model to track what the system believes a student knows. The domain model will usually include details to identify both correct and incorrect problem solutions. The student model often consists of knowledge components (KCs) and some number or probability representing the percent chance that the student has mastered the KC (Anderson, 1982). Tracking changes in student knowledge is called Knowledge Tracing in the cognitive tutors. This knowledge tracing can be a simple as counters of problems correct and incorrect or more complex cognitive models that use Bayesian techniques modeling probabilities of slips and guesses (Chang et. al, 2006).

2.2.1 Cognitive Tutors

The cognitive tutors based on ACT-R theory (Taatgen, and Anderson 2008), which were originally developed at Carnegie Mellon University, have the largest user base and have collected the most publicly available data including over 25 million student actions as of June 2009 (available at www.learnlab.org). ACT-R models knowledge by assigning production rules to model problem-solving steps. While students work in a cognitive tutor, problem-solving steps are matched to production rules in a process called model tracing. Each production rule has been hand tagged with individualized feedback for incorrect steps, and when a student solution is matched to a production rule, this feedback is given to the student. It is the creation of production rules and their accompanying annotations that take the greater part of time needed to construct cognitive tutors. The first cognitive tutor was a tutor to teach LISP programming (Anderson and Skwarecki 1986). Students using the LISP tutor completed problems 30% faster and scored 43% better than the non tutored students (Anderson et al 1985). Since then a number of successful tutors based on the ACT-R cognitive architecture have been created including the PAT tutor for algebra (Koedinger and Sueker 1996), the geometry tutor (Matsuda and VanLehn 2004), and the Ms. Lindquist web based algebra tutor (Heffernan and Koedinger 2002). The PAT tutor and geometry tutor were extensively studied in the classroom and shown to be effective (Koedinger et al 1997). These tutors became the basis of the spin off company called Carnegie Learning (www.carnegielearning.com), which currently provides full curriculms integrated with the intelligent tutors and is used by over 500,000 students per year. The Ms. Lindquist tutor is available free online for use in algebra, and is important to our work in the way it implements scaffolding. The Ms. Lindquist system implements scaffolding in the form of hints. Several hints per step are given followed by a "bottom out" hint that gives the student the answer to the particular problem or step. We followed this approach in the implementation of our methods in the Deep Thought logic tutor (Chapter 5 and 9). Because of the success of the cognitive tutors and the scaffolding strategy in Ms. Lindquist, we are adopting the ACT-R model as a basic framework, and our methods seek to achieve model tracing and provide scaffolding hints while students work problems.

2.2.2 Constraint Based Tutors

Another successful type of intelligent tutors are Constraint Based Tutors. These tutors do not track a student's path through a particular problem and are only concerned with the student's current state. Within this state, constraint-based tutors look for violations of problem constraints, requiring less time to construct and work well for less procedural problems (Mitrovic and Martin 2002). This work is based on Ohlsson's work on knowledge representation as constraints at the Univeristy of Illinois (Ohlsson 1994) and the ITS Research group in New Zealand led by Mitrovic. A typical constraint takes the form of: If <relevance condition> is true, then <satisfaction condition> had better also be true, otherwise something has gone wrong. One of the constraint based tutors is the SQL Tutor (Mitrovic and Martin 2002). An example constraint for this tutor is: If <Table name exists in SELECT Clause> is true, then <Table name exists in FROM Clause> had better also be true. If this constraint is not met in a student attempt, the tutor would let them know, meaning that if a column is selected from a specific table the table must be referenced. This tutor has been extensively used and today is supplied as a free web based supplement to Addison Wesley's SQL textbooks since 2006. Studies of the SQL tutor have shown that students using the tutor scored 11.5% points higher than those not using the tutor, and this result was statistically significant (t= 2.68, p = .01) (Mitrovic and Ohlsson 1999).

Although they have been shown to be effective, constraint based tutors suffer from several limitations. First, unlike a cognitive tutor that matches production rules at any state, constraint based tutors can only give feedback on what is correct in current problem state based on the what is known in the final solution state. If part of the current state is unknown to the constraints the tutor assumes it is correct while a cognitive tutor would assume the unknown portion of the state is incorrect. This means that constraint based tutors can only provide condition violation feedback, but may not be able to direct a student from their current state to the goal, as a cognitive tutor is built to do.

In a comparison of cognitve and constraint based tutors the KERMIT SQL constraint based tutor was rewritten using production rules like a cognitive tutor (Mitrovic et al 2003). The constraint based tutor was quicker to build but the feedback it provided to students was less specific (Mitrovic et al 2003). With these issues in mind, there are certain domains where constraint based tutors work well (such as formulating SQL queries), and other domains where other approaches are preferable. We implement our method in the logic domain. Since a solution is always possible from a given state if the problem is solvable, a constraint-based logic tutor would have a difficult time providing any strategies for solving the problem. Our method shrinks the overall solution space based on paths that previous students used to solve the problem, and seeks to provide hints like those in a cognitive tutor, to guide students to the solution.

2.2.3 Authoring Tools

As stated previously, one major problem with the adoption of ITSs into mainstream education is the difficultly in creating them. The major thrust so far to alleviate this difficulty has been to create authoring tools to simplify ITS creation.

One of the first successful authoring tools is REDEEM, which was built to reduce the time needed to create an ITS and allow teachers to apply their own teaching strategies in an existing computer-based training (CBT) system, in a variety of domains from mathematics to biology to learning how to operate machinery. The authoring tool has been used in secondary education (Aisworth et al 2000) and also by the Royal Navy (Ainsworth et al 2001). REDEEM has been shown to be more effective than a non-expert human tutor in improving student test scores (Ainsworth 2003), but does not have a detailed model of the domain like the more complex production rules systems. REDEEM instead focuses on improving the student-tutor interactions in order to increase student involvement with the tutor. REDEEM claims to not be as effective as cognitive tutors with complex production rules systems (Ainsworth 2004).

Another tool, the Cognitive Tutor Authoring Tool (CTAT), is based on production rules and ACT-R. CTAT has both a flash and java interface and can be used to quickly design a tutor interface. In addition to providing tools to build the tutor interface, CTAT allows users to create production rules that can be applied to specific problems. CTAT can also automatically extract production rules based on worked examples (Koedinger et al. 2004). Finally, CTAT allows teachers to work problems, in the interface that they have created, and annotate each step with hints and feedback messages. In this way, they predict frequent correct and incorrect steps in student solutions, and then annotate individual steps with appropriate hints and feedback in a behavior recording tool seen in figure 2.2. This is similar to the method we use in Chapter 9, although our method is automated. A variety of tutors have been created with CTAT, including a middle school math tutor (Koedinger et al. 2004).

The authors of the constraint-based tutors created the ASPIRE authoring tool to speed the development of constraint based tutors (Mitrovic et al 2006). ASPIRE provides an interface for the creation of constraints that can be applied to one or more problems. It has been used to create the current version of the SQL tutor and is the primary tool being used to create constraint based tutors. The SQL tutor developed using ASPIRE was shown to be effective in four separate studies where students using the constraint based tutors outperformed students not exposed to the tutor (Mitrovic et al 2007).

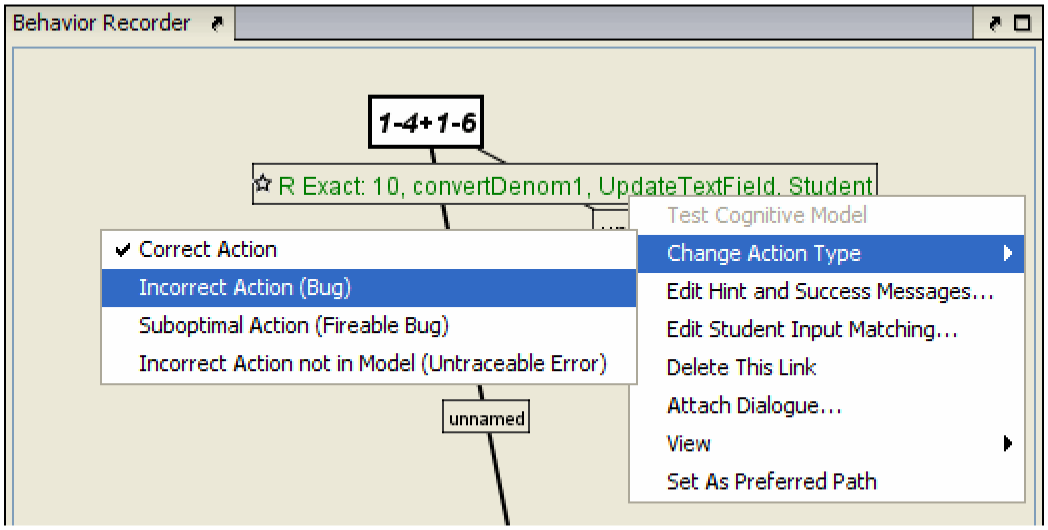

Behavior Recorder
1-4+1-6
R Exact: 10, convertDenom1, UpdateTextField, Student
Test Cognitive Model
Change Action Type
Edit Hint and Success Messages...
Edit Student Input Matching...
Delete This Link
Attach Dialogue...
View
Set As Preferred Path
✓ Correct Action
Incorrect Action (Bug)
Suboptimal Action (Fireable Bug)
Incorrect Action not in Model (Untraceable Error)
unnamed

Figure 2.2: An example of the CTAT behavior recorder interface. The white rectangle shows the student problem step. On this screen, a teacher is annotating the step to mark it wrong. Other options allow for adding hint and success messages as shown on the selection box.

Although these authoring tools provide a quicker way to develop ITSs, there are two major hurdles to their mainstream acceptance in the CAI field. First, each of these tools has a significant learning curve that requires developers to devote time that many would prefer to spend creating their own software. Second, since these tools are all academic based they lack the polished qualities of off the shelf software and in practice require assistance from the authoring tool team to create a finished product. Because of these hurdles authoring tools have not lived up to the promise of providing quick and easy adaptive features.

2.2.4 ITSs for logic

Our work with tutors has primary focused on the domain of solving propositional logic proofs. There are several other tutors that have been created for logic including

Carnegie Mellon Proof Tutor (CPT) (Scheines and Sieg 1994) and the Logic-ITA (Intelligent Teaching Assistant) (Merceron and Yacef 2005). Of these, the Logic-ITA is the most intelligent, verifying proof statements as a student enters them, and providing feedback after the proof is complete on student performance. Logic-ITA also has facilities for considerable logging and teacher feedback to support exploration of student performance (Merceron and Yacef 2005), but does not offer students help in planning their work. In the Logic-ITA tutor, student data from the log files of past classes was mined to create counter examples that showed student errors that were likely to be made on problems that used specific groups of rules. These examples were added to the tutors and incorporated into the lectures (Merceron and Yacef 2005). Logic-ITA as of this writing is no longer being used or available.

The CPT used production rules aimed at directing students down a correct solution path via a method the authors called “strategic thinking”. The goal was to limit the total search space that a student solving a proof could use. The CPT tutor is no longer used, but was the foundation for the Automated Proof Search (APros) proof tutor that is part of the online course offered by Carnegie Mellon (oli.web.cmu.edu/openlearning). APros implements a proof solver with the production rules from CPT that support strategic thinking (Sieg 2007). To date, there has not been an empirical study done on the effectiveness of APros.

In this research, we apply educational data mining from two existing logic tutors, the Proofs Tutorial and Deep Thought, to create domain models using MDPs to develop a method of model tracing to build problem specific cognitive models that can provide students context specific hints based on their current approach.

2.3 Using Data in Intelligent Tutors

In recent years the field of Educational Data Mining (EDM) has emerged from the Intelligent Tutoring and Artificial Intelligence in Eduction communities. EDM as a field uses raw data from educational systems to understand student learning and improve learning systems. In 2008, the 1st International Conference on Educational Data Mining was held in Montreal Canada, followed in 2009 with the 2nd International Educational Data Mining Conference held in Cordoba Spain. In one EDM approach, data from the ADVISOR reading tutor was used with reinforcement learning to build agents to represent students and the pedagogical model (tutor). These agents were used predict the amount of time students took to solve arithmetic problems, and to adapt instruction to minimize this time while meeting teacher-set instructional goals (Beck and Wolfe 2000). Another system called SimStudent uses data collected from CTAT to student models based on production rules (Matsuda et al 2007). Although SimStudent could be used like our methods to provide hints, it is being used to predict future student performance for the purpose of evaluating CTAT tutors. When SimStudent was trained on 15 problems, the system correctly predicted correct student responses over 80% of the time, but the current implementation of SimStudent does not correct student errors.

Student data has been used with CTAT to build initial models for an ITS, in an approach called Bootstrapping Novice Data (BND) (McClaren et al. 2004). In this approach, each student attempt from student log data is read in through a behavior recorder and then added to a large graph. The graph includes information on whether a path is good or bad based on information provided from the log data. Instructors can then view these graphs and add hints and feedback using the example tracing tutor interface in

CTAT. Although the BND approach saves time in entering example problems, it still requires expert instructors and programmers to create a tutor interface and annotate the extracted production rules with appropriate hints. Similar to the goal of BND, we seek to use student data to directly create student models for an ITS. However, instead of feeding student behavior data into CTAT to build a production rule system, our method uses data to generate Markov Decision Processes that represent all student approaches to a particular problem, and use these MDPs directly to generate hints. We further apply reinforcement learning via value iteration to the MDPs to give a specific value to each student state. This value incorporates knowledge from all past student attempts on the specific problem and can be used to direct student to best next state for solving the problem. Our work with Bayesian Knowledge Bases (Chapter 10) breaks our MDPs into their constituent parts to provide more general problem-solving steps that can be used for problems where no data are available.

2.4 Summary

Although ITSs have been shown to improve student learning, their use has not caught on at the same level of CAI in the classroom. The main reason for this is the difficulty in building these adaptive systems. Research has shown that it takes developers, educators, and subject experts 100-1,000 hours to build just one hour of adaptive instruction for a ITS (Murray, 1999). Adoption into schools is difficult for a number of reasons cited by Koedinger, et al (1997). One is that ITSs are self-contained with many of the teaching decisions already built in, meaning that in order to use these systems educators are locked in to using the approaches they give. Distribution and support of the software are important issues. Many tutoring systems are available, but are not well documented or supported. Installing and using them can be difficult. A number of tutors like Ms. Lindquist are available online to make access easier (Heffernan and Koedinger 2002). However, approaches are still needed to make new tutors and make them so that teachers can use them as they wish.

In this thesis we have shown that a Markov Decision Process derived from data can be used to represent procedural domain knowledge in several domains, that we can combine this MDP with instructor knowledge to automatically generate effective, context-specific hints adapted to individual students, and that applying a Bayesian Knowledge Base to these MDPs across multiple problems can be used to create hints for problems where no data yet exists. We believe that our work provides a basis for adding more adaptive support for students to CAI. Our work also provides a foundation for building a tool for teachers to understand student work and modify hints and feedback to adapt a tutor for their own uses.

CHAPTER 3: MARKOV MODELS FOR HINT GENERATION

In this chapter, we explore our novel application of Markov decision processes (MDPs) for building and visualizing student knowledge models, and integrating these models into existing computer-aided instructional tools to automatically generate context specific hints. The MDP method is straightforward to implement in many educational domains where computer aided instruction exists. We have focused on improving existing CAI to teach logic proofs. This is a domain with multiple solutions paths to most problems, and based on the rule set available it is likely impossible to cover all possible steps a student might take. For example, students might use the "addition" rule to create a new statement by adding new variables to an existing true statement, leading to infinite possible variations that students can derive.

In this chapter, we first demonstrate how to create a Markov model including defining the state and actions. From this Markov model, a learning algorithm is applied to generate a value for each state. Then, we visually explore student responses combined into a MDP that represents the variety of observed solutions to a problem. Next, we discuss how to construct a hint generator using the MDP and how to construct a feasibility study to determine the probability of hint availability. This feasibility technique is a valuable contribution that can be used to make decisions about the appropriateness of MDPs for hint generation in a particular problem domain. Finally, we describe how a MDP hint generator has been integrated into a logic tutor and discuss its effects on student learning.

3.1 Background

The initial goal for our method was to speed the extraction of production rules from student log files using Markov Decision Processes (MDPs). Since initial conception, we have modified our MDP approach to allow creation of hints without the intermediate creation of production rules. We also leverage existing computer-aided instruction (CAI) tools and data to reduce time to build an intelligent tutor, using the already-built CAI problem-solving interface along with student data to create MDPs for model tracing, and have reduced the hint annotation time by creating an automatic hint generator.

MDP-based tutors work best in situations where CAI already exists, problems consist of a series of related steps most often solved using similar strategies, and log data is readily available. Many CAI tools exist for math and science problems, which often have multiple related steps but involve a limited number of problem-solving strategies. For the best hint generation, it is best if errors and correct solutions are labeled or detected. However, the MDP approach can be used to analyze problems and build probabilistic MDP tutors without correct and incorrect labels.

3.2 Method

The process of creating an MDP analysis or tutor has several phases. In the first phase, we use log data to create a directed graph of student work. We convert each step of a student solution attempt to a state, and states are joined to each other by the actions a student took to move from one state (problem step) to the next. We combine all states and actions to create a graph of all student work, by taking the union of all states and actions, and mapping identical states to one another. In the second phase, we convert this graph of student work into a Markov Decision Process (MDP), where state-transition probabilities are set and reward values are assigned to each state-action pair. Once constructed, this MDP can be used to visualize all student work, with reward values in the MDP representing a measure of the value of each state in reaching a correct or common pattern in problem-solving.

As part of creating an automatic hint generator, we constructed a matching function that behaves as a model tracer that matches student work to states in the MDP. We also construct a hint template in collaboration with domain experts. Together, the MDP, hint template, and matching function combine to create the hint generator which provides individualized hints for a particular problem. When the hint generator is added to a CAI, we create an MDP-tutor.

Formally, a Markov decision process (MDP) is defined by its state set ***S***, action set ***A***, transition probabilities ***P***, and a reward function ***R*** (Sutton and Barto 1998) (Russell and Norvig 1995). On executing action ***a*** in state ***s*** the probability of transitioning to state ***s´*** is denoted $P(s'|s,a)$ and the expected reward associated with that transition is denoted $R(s'|s,a)$. States and transitions in the MDP must be constructed to adhere to the Markov property, which states that the probabilities of reaching future states depend only on the current state, regardless of past states. This means that states do not have memory, so they must contain all information, or features, needed to fully describe each step taken in a problem until the current moment.

The most difficult aspects of MDP problem-solving analysis are feature selection and granularity for defining state and actions, and assigning appropriate values that accurately reflect both successful and incorrect solutions. If a state contains too many features, such as every key or mouse press in a solution, it is likely that too many states will be created, and matching a students' current state to these states will be more difficult. Creating states that contain too few features could result in less specific hints. The state features can be collected in an ordered or unordered fashion. Ordered states retain information on how the student worked the problem and may be particularly important in some domains, but preserving order will increase the state granularity. Actions are the transitions between states. In the context of CAI, an action is anything that changes the problem state. Depending on how states and actions are defined it is possible and acceptable for the same action to lead to multiple different states. For example, when a rule such as DeMorgan's is applied to a logic statement, it could be applied to different parts of the statement, yielding different states. A rule/action might also be misapplied, giving incorrect states in one application and correct ones in another. Similar to state descriptions, the granularity of action descriptors determines whether this type of overlap can occur.

Figure 3.1 shows how we have applied our technique to turn the Deep Thought Logic CAI into an MDP-tutor, which provides hints while students solve logic proofs (Croy 1999). In Deep Thought, state features include the premises, statements, and conclusion visible on the screen. For example, the state in Figure 3.1 could be described as [not(T and L), if not T then not N, not (E or T), if N then T, not E and not T, not T, not N]. Logic tutor "actions" occur when a student clicks on a rule button, shown on the left in Figure 3.1. This granularity allows students to attempt partial steps without invasive feedback during their exploratory step construction.

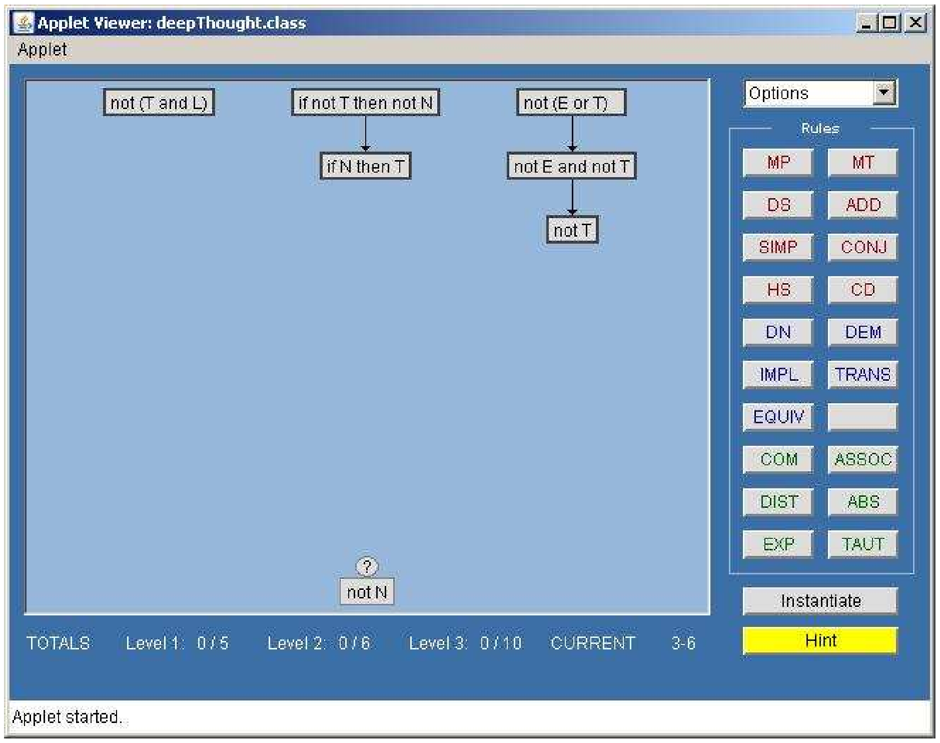


Figure 3.1: An example of the Deep Thought Interface, with problem 3.6 partially completed. The statements at the top are “premises” which are given at the problem start, and the student is trying to prove the statement “not N”, as denoted with a question mark.

It is important to consider errors and how they will be recorded into states. When a student makes an error during a problem attempt, a state should be created, but then it must be decided if the student will remain in the error state or will be moved back to the last non-error state. Often this decision will be made based on the way the CAI works when the student makes an error. Does the error remain on the screen, or is a message provided and the student is returned to the state before the error occurred? If errors are persistent in the CAI, they should remain part of the state. When the reward values are assigned, as described below, it is important to penalize only the states in which an error originally occurs, and not those that follow on. In Deep Thought, errors are detected and are not retained on the screen, so in our MDP, error states are recorded, but these states connect back to the preceding problem state.

Once all student attempts are combined into a single graph, we need a way to evaluate states in the MDP and find optimal solutions to complete a problem. In a traditional sense, an optimal solution is a shortest path that takes a student from the start to the correct completion of a problem. If we can determine values for whole student attempts, corresponding to leaves in the student work graph, then we can use reinforcement learning to assign values to intermediate problem states. The simplest way to assign values to valid solutions is to create an artificial "goal" state and connect the last state in a successful problem solution to this goal state.

We set a large reward for the goal state(s) and negative rewards for incorrect states. A small negative reward is assigned to each transition. Setting a negative reward on actions causes the MDP to penalize longer solutions. Next, value iteration, a reinforcement learning technique using Bellman backup, is used to assign values to all states in the MDP (Sutton and Barto 1998). The equation for calculating values ***V(s)*** for each state ***s***, where ***R(s)*** is the reward for the state, γ is the discount factor, and $\boldsymbol{P_a(s,s')}$ is the probability that action ***a*** will take state ***s*** to state ***s'*** is:

$$V(s) = R(s) + \gamma \max_a \sum_{s'} P_a(s,s')\, V(s')$$

For value iteration, ***V*** is calculated for each state until there is little change in the function over the entire state space. Once this is complete, the optimal solution in the MDP corresponds to taking a greedy traversal approach in the MDP (Barnes and Stamper 2007). The reward values for each state then indicate how close to the goal a state is, while probabilities of each transition reveal the frequency of taking a certain action in a certain state.

The actual reward values may need to be modified to fit a certain domain. In our tests with the logic tutor we initially used a large reward for the goal, +100, and a negative reward, or penalty of -10, for errors. As we analyzed the data it was apparent that our error penalty was too high and caused the best path through the MDP to reflect an error avoidance policy. To correct this issue a smaller error penalty can be used, which will give errors less weight and more weight to a shorter path. In order to determine the best policy to use for a particular domain and group of users it is important to work with domain experts and review several problem MDPs before using them in a live environment.

3.3 Hint Generator

Once MDPs are created they can be used with a matching function and hint template to automatically generate individualized hints. In order to prepare the MDP for hint generation, we first remove error states from the MDP, since we would never want to lead students to an error. At this point, when a hint is required the first step is to match the student's current state to a state in the MDP, and we describe several matching functions in Chapter 5. If found, the hint generator can compare that state's successor states, and use the successor with the highest reward value to generate a hint. The hint can suggest the action to take, the state features in the next state, or the state features in the current state that are used to get to the next state. Educators who are using the CAI will be invaluable in determining which state features and actions are important and how they can be generalized into a hint template. Typically a hint sequence will be provided for a given state. A hint sequence refers to hints that are all derived from the same current state, and would be sequentially given to the students each time the hint button was pressed while the problem remained in the same state. A hint sequence for the tutor might consist of four types of hints: 1) indicate a goal expression to derive from the next state, 2) indicate the next action, 3) indicate what parts of the current state are used to reach the next state, and 4) a bottom-out hint combining 1-3. For the problem state seen in Figure 3.1 from the Deep Thought logic tutor, the hint sequence given in Table 3.1 would be generated. These hints are based on the best next action and state from the student's current state. For each state, four distinct hints are generated. If a student requests a hint, then makes an error, and requests a hint again, the next hint generated is the next one in the current sequence. Once a student performs a correct step, the hint sequence is reset.

Table 3.1 Hint sequence derived from example student solution. The best successor state is decomposed into parts to generate each of the hints.

Hint #	**Hint Text**
1	Try to derive not N working forward
2	Highlight if not T then not N and not T to derive it
3	Click on the rule Modus Ponens (MP)
4	Highlight if not T then not N and not T and click on Modus Ponens (MP) to get not N

One additional consideration is what to do when no match to the current state exists in the MDP. In this case when a specific hint is not available the hint generator can either return that a hint is not available or the hint generator can always seek backwards in the student's path until a state is reached where a hint can be given.

Our hypothesis was that we could provide hints a majority of the time using just one semester of data to train an MDP (Barnes and Stamper 2008). In order to validate this hypothesis, we performed a cross-validation study on four semesters of data, checking to see how many hints were available using different semesters and different matching functions for hint generation. This analysis is shown in detail in Chapter 5. We have added a hint generator to the Deep Thought CAI to create an MDP-tutor for several Deep Thought problems, and in Chapters 6 and 7 we discuss two experiments to evaluate the generated hints in an actual classroom.

3.4 Summary

In this chapter we have shown how to use student data gathered from CAI to create MDPs to analyze problems and provide context-specific hints. The MDP method for generating hints was applied to a data from tutors for teaching logic proofs, but the method is straightforward to implement in many educational domains where computer aided instruction exists for problem solving in procedural domains. The implementation of our method on three additional domains is in fully shown in Chapter 11.

CHAPTER 4: MARKOV MODELS AS A VISUALIZATION TOOL

Once a MDP is generated, it can be used by educators to provide a visualization of student solutions to a problem. These visualizations show the many different ways in which students solve a problem. In this chapter we test the following hypotheses:

> Hypothesis 1: The generated MDPs can be useful to educators to identify student strategies for solving a problem.
>
> Hypothesis 2: Educators will be able to better identify trouble spots using the MDPs.

Each step of the problem is represented by a state and a reward value that reflect the step's frequency, correctness, and nearness to the solution. This visualization allows educators insight into student learning, revealing correct approaches and student behaviors that can indicate areas for further instruction. A sample problem is shown in Table 4.1 and Figure 4.1, and the visualization of the most frequent valid states in the MDP created for this problem is given in Figure 4.2. The key to the states and actions in this graph are explained in Table 4.2. Figure 4.2 shows the aggregate MDP restricted to only valid states and frequent state-action pairs. Without knowing any information about the domain for the example, the graph still shows interesting information. An observer can see that there is only one frequent successful path – and this solution is also a concise, expert solution. Very quickly, educators can identify the strategies most students are using to solve a problem, and they can also identify paths in which students appear to be having trouble solving the problem.

4.1 Background

In this chapter, we explore all student attempts at proof solutions, including partial proofs and incorrect rule applications, and use visualization tools to learn how this work can be extended to automatically extract a production rule system to add to our logic proof tutorial. In our first work (Stamper 2006), we performed a pilot study to extract Markov decision processes for a simple proof from three semesters of student data from Deep Thought, and verified that the rules extracted by the MDP conformed to expert-derived rules and generated buggy rules that surprised experts. In this chapter, we apply the technique and extend it with visualization tools to new data from the Proofs Tutorial.

The Proofs Tutorial is a computer-aided learning tool implemented on NovaNET (http://www.pearsondigital.com/novanet/). This program has been used for practice and feedback in writing proofs in university discrete mathematics courses taught by several instructors at North Carolina State University since 2002. In the Proofs Tutorial, students are assigned a set of 10 problems that range from simpler logical equivalence applications to more complex inference proofs. (The tutorial can check arbitrary proofs, but it is used for a standard set of exercises). In the tutorial, students type in consecutive lines of a proof, which consist of 4 parts: the statement, reference lines, the axiom used, and the substitutions which allow the axiom to be applied. After the student enters these 4 parts to a line, the statement, reference lines, axiom, and substitutions are verified. If any of these are incorrect, a warning message is shown, and the line is deleted (but saved for later analysis).

Table 4.1 lists an example student solution. Figure 4.1 is a graphical representation of this proof, with givens as white circles, errors as orange circles, and premises as rectangles.

Table 4.1: Sample Proof 1 Solution (red lines are errors)

Statement	Line	Reason
1. a → b		Given
2. c → d		Given
3. ¬ (a → d)		Given
¬ a v d	**3**	**rule IM (error)**
4. a ^ ¬ d	3	rule IM implication
5. a	4	rule S simplification
b	**4**	**rule MP (error)**
b	**1**	**rule MP (error)**
6. b	1,5	rule MP modus ponens
7. ¬ d	4	rule S simplification
8. ¬c	2,7	rule MT modus tollens
9. b ^ ¬c	6,8	rule CJ conjunction

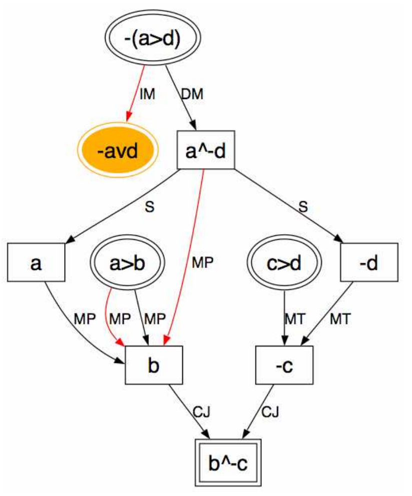


Figure 4.1: Graph of Proof 1 Solution (Red lines are errors)

4.2 Experiment

The experiment presented in this chapter uses data from the four fall semesters of 2003-2006, where an average of 120 students take the discrete math course at NC State

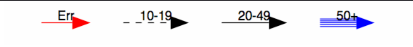
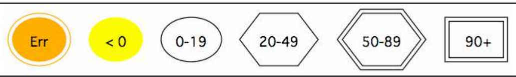
University each fall. Students in this course are typically engineering and computer science students in their second or third year of college, but most have not been exposed to a course in logic. Students attend several lectures on propositional logic and complete an online homework where students complete truth tables and fill in the blanks in partially-completed proofs. Students then use the Proofs Tutorial to solve 10 proofs as homework, directly or using proof by contradiction. The majority of students used direct proof to solve proof 1. We extracted 429 of students' first attempts at direct solutions to proof 1 from the Proofs Tutorial. We then removed invalid data (such as proofs with only one step to reach the conclusion), resulting in 416 student proofs. Of these, 283 (70%) were complete and 133 (30%) were partial proofs. Due to storage limitations, a few (6) of these proofs may have been completed by students but not fully recorded. The average lengths were 13 and 10 lines, respectively, for completed and partial proofs. This indicates that students did attempt to complete the proof.

After cleaning the data, we load the proofs into a database and build an MDP for the data. We then set a large reward for the goal state (100) and penalties for incorrect states (10) and a cost for taking each action (1). Setting a non-zero cost on actions causes the MDP to penalize longer solutions (but we set this at 1/10 the cost of taking an incorrect step). These values may need to be adjusted for different sizes of MDPs. We apply the value iteration with a Bellman backup (reinforcement learning technique) to assign reward values to all states in the MDP.

As previously explained in Chapter 3, we use value iteration to generate a value, $\boldsymbol{V}$, that is calculated for each state until there is little change in the value function over the entire state space. Once this is complete, the optimal solution in the MDP corresponds to taking a greedy traversal approach in the MDP (Sutton and Barto 1998). The rewards for each state then indicate how close to the goal a state is, while probabilities of each transition reveal the frequency of taking a certain action in a certain state. For generating a visualization, just one step of value iteration was performed to cascade goal values through the MDP.

We created MDPs for each semester of data, and one for the aggregate. This resulted in a set of states, reward values, and actions for each MDP. On average, the four semesters yielded MDPs with158 states (ranging from 95-226 states in a semester). The most frequent approaches to problems were very similar across semesters, and the most frequent errors were also repeated. Therefore, in the remainder of this paper, we examine the aggregate MDP. Using Excel®, we assigned labels to each state in the MDP (just using the latest premise added), colors for errors, state values, and action frequencies, and prepared the data for display. We used GraphViz (www.graphviz.org) to display the output and convert into pictures. Table 2 shows the legend for nodes and edges. After graphing each MDP, we continually refined the data being displayed to explore questions about the student data. We present our findings in the following section.

Table 2: Legend for MDP edges and nodes

Edges (Values=Frequency)	Nodes (Values=Rewards)
	

4.3 Results and Discussion

The aggregate MDP run on all four semesters of data has a total of 547 states (individual semester MDPs each contained 95-226 unique states). The MDP can be seen in Appendix B. From interactive exploration, the course instructor found that 90% of all student errors related to explaining their actions, and a great majority of these were on the simplification rule. This indicates a need to improve the interface for students to perform this explanation. The instructor also found that students commit a great number of errors in the first step but less as they progress, indicating that students have the most trouble getting started but after that don't get stuck as often as the instructor thought they might. This work provides the foundation for building a tool for teacher visualization that will allow for pruning nodes below a certain reward or frequency, and also for highlighting all the correct or incorrect applications of a particular action. We demonstrate some of these views in Figures 4.2-4.3.

Figure 4.2 shows the aggregate MDP restricted to only valid states and frequent state-action pairs. Shaded or yellow nodes indicate states where students were likely to make errors in the next step. The graph shows only one frequent successful path – indicating an optimal solution that students can perform. This path also corresponds to an expert solution. On this path, it seems that errors are occurring in going from state (a^-d) to (a), demonstrating our observation that applying simplification is difficult for students. As in the overall MDP, this restricted view also shows many errors at the start, indicating that students have trouble getting started. Following the shaded path, second from the lowest in Figure 4.2, we observe that several (20-49) students apply IM (implication) in various ways; this path is very error-prone and does not (frequently) lead to a solution. This indicates a need to help students plan proof strategies. In future work we could detect this type of path in the MDP and offer hints to help avoid it.

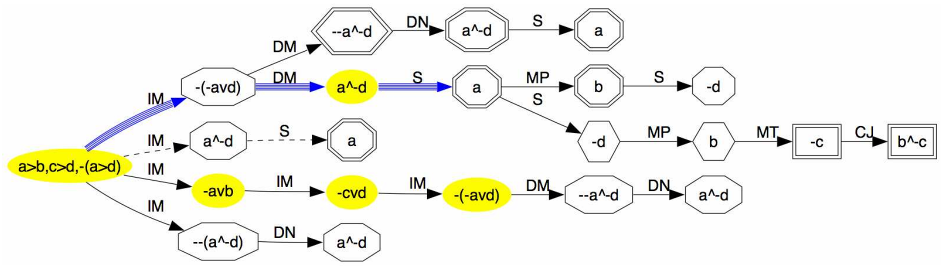


Figure 4.2: View of MDP restricted to valid states and frequent actions

To further examine student approaches, we expand this graph to include error states in Figure 4.3. In most errors, students find the correct premise (e.g. –d, which is correct) but have trouble explaining how the rule was applied to get the premise. For example, the rule for S (simplification) states (p^q)→p, so to obtain (a^-d)→-d the student must show that we substitute p=-d and q=a. From this information, the course instructor could conclude that students need a better interface for explaining rule applications. For example, in another logic tutor called Deep Thought, students are not required to perform substitutions if the program can detect them itself, and for simplification, the program asks: Right or Left? Anecdotally and intuitively, students seem to have less trouble with this approach. All but two of the remaining error states (indicated with darker shading and a double border) are due to substitution errors. (Start) → (–(a^-d)) is an incorrect application of IM (it's missing a not), while Contrapositive (CP) was frequently applied to obtain –c from c>d and –d (which needs Modus Tollens, MT). To course instructors, these findings indicate that more practice might be needed with these rules in the context of negated variables.

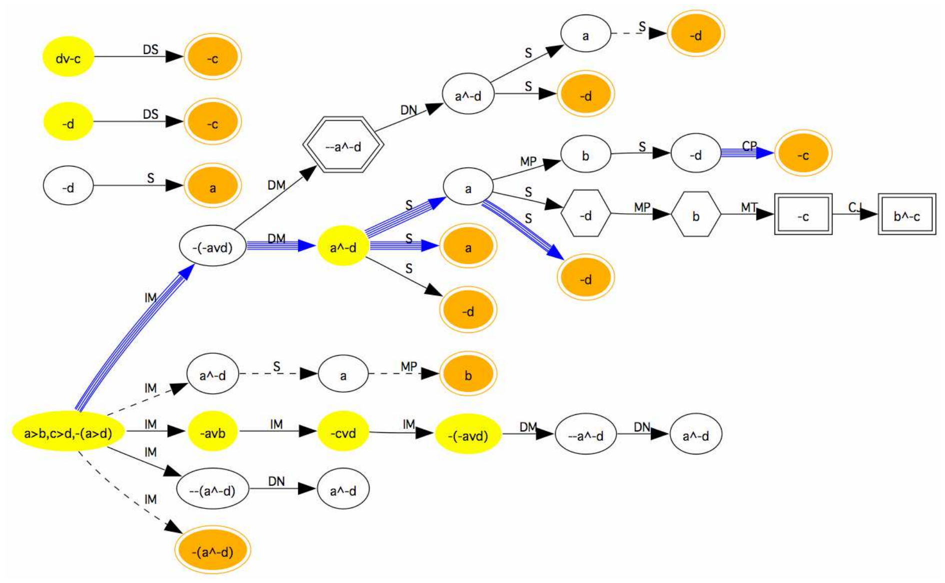


Figure 4.3: View of MDP restricted to frequent states and actions

These visualizations are useful in understanding what states will be used to generate hints based on frequent responses that are closest to the problem solution. However, this does not yield insight into a more general application of MDPs to proofs and what types of production rules might be generated for general problems. To make more general production rules for proof problems, we will take MDPs from several problems and attempt to learn general rules. For instance, a production rule often applied by experts is, “if (p $\rightarrow$ q) and (p) are statements then apply MP to obtain the new statement (q)”. One solution to this issue is to break MDP states down into their problem steps, which can be seen in Chapter 10.

We created Figure 4.4, (which is a directed graph, not a MDP), by mapping all correct states with a common last statement into a single node, which corresponds to grouping unique action/statement pairs. We then eliminated all low-frequency actions (taken less than 9 times) and the simplification rule (since we plan to change the interface to improve usage of this rule). This new visualization of the paths of student solutions allows us to track frequent solution attempts regardless of order. In other words, our previous MDP views correspond to unique paths, while this graph shows us relationships between consecutive steps in the proof regardless of what was done before.

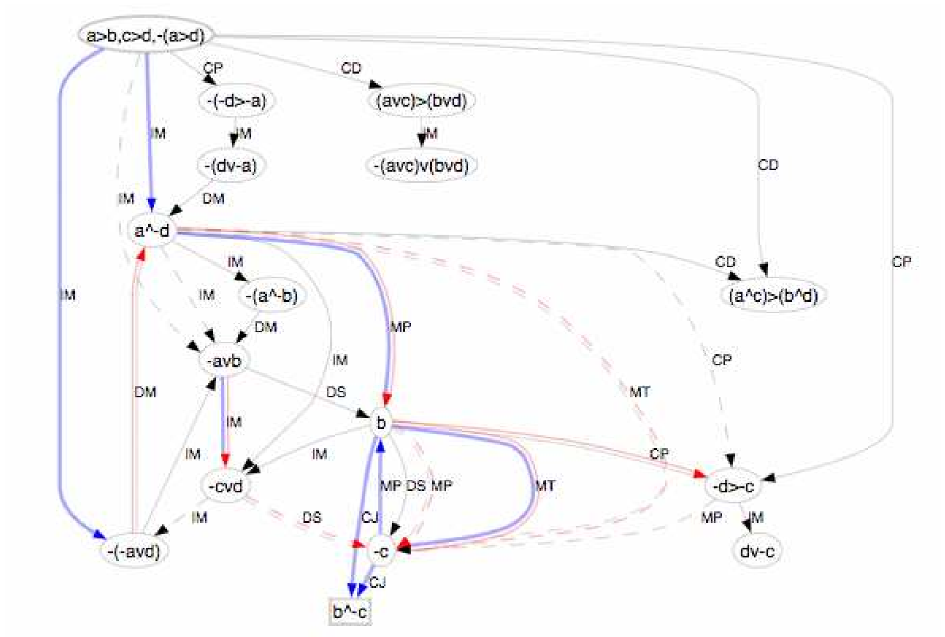


Figure 4.4: Graph showing inferences, unique premises and frequent (>9) actions

In Figure 4.4, there are some dead-end paths, meaning that several students started proofs in these directions but few students were able to reach the solution this way. These dead-end paths can be used to derive feedback to students that their approach may not be productive. We can also use Figure 4.4 to derive most frequent orderings of student solutions. To do so, we start at the (top) start node and choose a frequent (wide) edge, and repeat without visiting nodes twice, until we reach the solution, as in Figure 4.5.

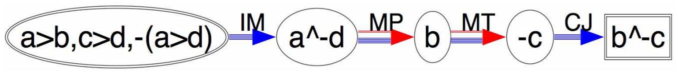


Figure 4.5: Most frequent proof solution sequences, derived from Figure 5

Some secondary approaches are shown in Figure 4.6, demonstrating how a teacher could use Figure 4.4 by following particular paths but not repeating any nodes visited, to find unique solutions to the proof. These approaches demonstrate students' frequent preference to use Disjunctive Syllogism (DS), even though these solutions are longer.

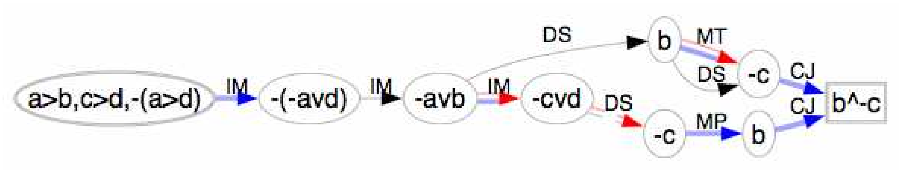


Figure 4.6: Secondary proof approaches derived from Figure 5

In both Figures 4.5 and 4.6, there are errors (on the double edges) leading to and from states containing negated variables. This observation reflects instructors' experience that students need more practice in using rules when variables are of negated, and in applying rules in sequence to achieve a goal.

4.4 Summary

This chapter describes the basis of our approach to mining Markov decision processes from student work to automatically support model tracing and discussed how this approach can be applied to a particular computer-aided instructional system. This approach differs from prior work in authoring tutoring systems by mining actual student data, rather than relying on teachers to add examples the system can learn from. In this chapter, we have explored visualizations of the Markov decision processes extracted from solutions to a formal logic proof to determine how to improve the tutor and how we might proceed in building useful production rules and feedback. We believe that the process we have applied to creating problem visualizations can be useful in learning about other problem-solving processes from student data, instead of creating expert systems by hand. From these visualizations, instructors learned:

1) Although instructors hypothesized that students need help in planning, this did not seem to be the case. Instead, students needed help on getting started.

2) As expected, students need more practice with negation.

3) The overwhelming majority of student errors were in explaining rule applications. A better interface is needed for this step.

We have also concluded that the extracted MDPs will be useful in generating student feedback. The extracted MDP does contain a frequent expert-like path and contains a significant number of correct paths and student errors. The proofs and Depp Thought tutors can already classify many errors students make. Adding the MDP to these tutors will enable them to provide hints. This MDP can constantly learn from new student data. We note that on cold start for a new problem that has no student data, the system will still act as a problem-solving environment, but after even one semester of data is collected, good feedback can be generated as shown in Chapter 5. As more data is added, more automated feedback can be generated.

CHAPTER 5: AUTOMATIC HINT GENERATION VALIDATION

The goal of this chapter is to use historical data to verify that we can generate a sufficient number of hints to support students while working problems in the the NCSU Proofs Tutorial. Our hypothesis is:

> Hypothesis: We will be able to generate hints, from one semester of data, which would have been available 50% of the time if past students could have had hints on every problem step they took.

The hints we generate will help students focus their attention on an appropriate next sub-goal.

5.1 Background

Giving students hints has been shown to improve learning and skill transfer over minimal and condition violation feedback (McKendree 1990). Since the Proofs Tutorial has been used for several years, we have a large corpus of data to use in building student models from historical data. We create a student model for each problem, and use it to generate intelligent hints. As a new student works a problem, we record his or her sequence of actions as a state. If the current state is matched in the model, and the matched state has a successor closer to the goal, we will enable a Hint Button, as discussed in the next chapter, to give contextual help. From the successor state with the highest reward value, we derive a hint sequence: 1) indicate a goal expression to derive, 2) indicate the rule to apply next, 3) indicate the premises (lines) where the rule can be used, and 4) a bottom-out hint combining 1-3. An example of this hint sequence was previously described in Table 3.1. If a student's state is not found in the model, the Hint Button will be disabled. Such a student will not get goal feedback. However, we can add the student's action and its correctness to our database, and periodically run value iteration to update the reward function values. Before an update is applied, we could test the update by examining new MDP states to ensure that unusual solutions have not superseded existing good solutions as hint sources.

The method of automatic hint generation using previous student data we propose in this chapter reduces the expert knowledge needed to generate intelligent, context-dependent hints. The system is capable of continued refinement as new data is provided. In this chapter, we demonstrate the feasibility of our hint generation approach through simulation experiments on existing student data. We show that our approach is appropriate for generating hints for specific problems with existing prior data, while we show in Chapter 10 that machine learning applied to MDPs may be used to create automated hints for new problems in the same domain.

In this chapter, we use historical data to estimate the availability of hints using different types of state-matching functions and differing datasets for training. We use data from the four fall semesters of 2003-2006 (denoted f3-f6), where an average of 220 students take the discrete math course each fall. Students in this course are typically engineering students in their 2nd or 3rd years, but most have not been exposed to a course in logic. Students attend several lectures on logic and then use the Proofs Tutorial to solve 10 proofs. Sixty percent of students used direct proof when solving proof 1. We extracted 537 of students' first attempts at direct solutions to proof 1.

The data were validated by hand, by extracting all premises generated by students, and removing those that 1) were false or unjustifiable, or 2) were of improper format. We also remove all student steps using axioms Conjunction, Double Negation, and Commutative, since students are allowed to skip these steps in the tutorial. After cleaning the data, there were 523 attempts at proof 1. Of these, 381 (73%) were complete and 142 (27%) were partial proofs, indicating that most students completed the proof. The average lengths, including errors, were 13 and 10 steps, respectively, for completed and partial proofs. When excluding errors and removed steps, the average number of lines in each student proof is 6.3 steps. The validation process took about 2 hours for an experienced instructor, and could be automated using the existing truth and syntax-checking program in our tutorial. We realized that on rare occasions, errors are not properly detected in the tutorial (less than 10 premises were removed).

We performed two experiments to explore the capability of our method to generate automated hints. In each experiment, we isolated the data into training and test sets, where the training set was used to generate the Markov Decision Process (MDP) as described in Chapter 3, and the test set was used to explore hint availability. The process for comparing the test set to the MDP consists of several steps. Because of the structure of the tutorial, we first removed all error states from the MDP and from student attempts before comparison, since the tutorial provides error messages and deletes the corresponding error from the student proof. Then, each attempt in the test set is mapped onto a sequence of states. For each test state, there are two requirements for a hint to be available: 1) there must be a "matching" state in the MDP, and 2) the "matching" state must have a successor state in the MDP (i.e. it cannot be a dead end). The closer the match between a test state and the corresponding MDP state, the more context-specific the hint based on that match will be.

To maximize the probability that our generated hints are in line with a student's current strategy, we seek to give hints based on states very similar to the current state. We considered four matching functions: 1) ordered (exact), 2) unordered, 3) ordered minus the latest premise, and 4) unordered minus the latest premise. An ordered, or exact, state match means that another student has taken the same sequence of steps in solving the proof. An unordered state match means that there is a state with exactly the same premises, but they were not necessarily reached in the same order. An "ordered-1" match looks for an exact match between the student's previous state and an MDP state. An "unordered-1" match looks for an unordered match between the student's previous state and an MDP state. Once a match is made, we generate a hint using the optimal successor state from the matching state. The more specific the match, the more contextualized the hint. Hints generated using unordered matches will reveal steps taken by other students in the same problem state, but who might be using a different approach to problem solving, so these hints may differ from hints based on ordered matches.

To determine hint availability, we calculated two numbers for each match type. The first is "move matches": the percentage of test states, or "moves", including duplicates, with matches in the MDP. The second is the "unique matches": the percentage of unique test states that have matches in the MDP. These move matches give us a measure of the probability that a hint is available for each move. Unique matches reflects the percent overlap in test and training sets, and could indicate if one class is particularly different from the training set.

5.2 Experiments

We performed two experiments to test the hypothesis that the MDP method can be used to provide hints on at least 50% of the steps students took on solving this problem in our historical data sets using just one semester of student data. The first experiment compares several semesters of data by identifying the number of overlapping states created which shows how often a hint would be available. In the second experiment we explore how much data is needed before hints can be reliably provided by simulating how continuous updates of our MDP would affect hint availability.

5.2.1 Comparing classes

In this experiment, we explored the ability of our system to provide hints using one, two, three, or four semesters of data to build MDPs. Similar to a cross-validation study, each semester is used as a test set while all the remaining semesters are used as training sets for MDPs. This experiment provides us insight into the number of semesters of data we might need to provide hints a reasonable percentage of the time while students are solving proofs. Table 5.1 presents the data for each semester. Semester f5 was unusual: there were a small number of states, but a large number of moves, suggesting that students solved this proof in very similar ways.

Table 5.1. Semester data, including attempts, moves, and states in the MDP for each semester

Semester	# Attempts	MDP states	# Moves
f3	172	206	711
f4	154	210	622
f5	123	94	500
f6	74	133	304

We hypothesized that we could provide hints a majority of the time (e.g. at least 50% of the time) using just one semester as our MDP training data. Table 5.2 shows the percent ordered matches between each semester and the remaining combinations of training sets. We were very encouraged by these data, suggesting that our system would provide highly contextualized hints over sixty-six percent of the time, in the worst case, after just one semester of training. In all cases, adding more data increased the probability of providing hints, though we do see diminishing returns when comparing the marginal increase between 1-2 (6.8%) and 2-3 (2.8%) semesters of data.

Table 5.2. Avg % move matches across semesters using the ordered test sets and MDPs

Test set	1-sem. MDPs	2-sem. MDPs	3-sem. MDPs
f3	68.73%	75.67%	78.62%
f4	69.77%	77.71%	81.03%
f5	86.33%	90.80%	92.00%
f6	66.34%	74.12%	77.63%
Average	**72.79%**	**79.57%**	**82.32%**

Table 5.3. Avg % move matches across semesters using the unordered test sets and MDPs

Test set	1-sem. MDPs	2-sem. MDPs	3-sem. MDPs
f3	76.62%	82.16%	84.37%
f4	75.35%	81.99%	84.41%
f5	91.93%	94.40%	95.40%
f6	74.56%	82.35%	84.87%
Average	**79.62%**	**85.22%**	**87.26%**

Table 5.4. Avg % move matches across semesters using the ordered-1 test sets and MDPs

Test set	1-sem. MDPs	2-sem. MDPs	3-sem. MDPs
f3	76.92%	85.14%	89.00%
f4	76.26%	85.69%	90.35%
f5	90.78%	96.19%	97.80%
f6	75.55%	84.32%	89.14%
Average	**79.88%**	**87.84%**	**91.57%**

Table 5.5. Avg % move matches across semesters using the un<u>ordered-1</u> test sets and MDPs

Test set	1-sem. MDPs	2-sem. MDPs	3-sem. MDPs
f3	82.63%	89.19%	91.99%
f4	81.73%	90.14%	93.41%
f5	94.60%	97.00%	98.00%
f6	81.03%	89.69%	92.43%
Average	**85.00%**	**91.50%**	**93.96%**

Our experiments using the remaining matching techniques (unordered, ordered-1 and unordered-1) showed consistent increases going from 1-semester MDPs up to 2-semester MDPs as seen in Table 5.3, 5.4, and 5.5. However, the increases between 2- and 3-semester MDPs are decreasing, suggesting consistent diminishing returns for adding more data to the MDPs. Table 5.6 lists the average percent matches for each of our experiments using the four matching functions. This table gives an indication of the tradeoffs between using multiple semesters of data versus multiple techniques for matching. Here, we see that, on average, for 72% of moves, we can provide highly contextualized (ordered) hints using just one semester of data. With two semesters of data, we can provide these hints almost 80% of the time, but this only increases to 82% for three semesters of data. If we wished to provide hints after collecting just one semester of data, we could provide less contextualized hints for those who don't have ordered matches in the MDP. There is a nearly identical increase in the match rate, to almost 80%, by providing hints using either unordered or ordered-1 searches. We can provide hints an additional five percent of the time if we add the unordered-1 match function.

Table 5.6. Comparison of % move matches across multiple semesters and matching techniques

Matching	1-sem. MDPs	2-sem. MDPs	3-sem. MDPs
Ordered	72.79%	79.57%	82.32%
Unordered	79.62%	85.22%	87.26%
Ordered-1	79.88%	87.84%	91.57%
Unordered-1	85.00%	91.50%	93.96%

Table 5.7. Comparison of % move matches, excluding f5 from all sets

Test set	1-sem. MDPs	2-sem. MDPs
Ordered	70.97%	78.05%
Unordered	78.69%	83.59%
Ord-1	79.02%	87.99%
Unord-1	85.77%	91.86%

When analyzing these data, we observed a skew in all statistics because of the unusual distribution of states and moves in f5. We therefore repeated all experiments excluding f5, and the results are given in Table 5.7. The differences caused by skew in f5 had a smaller effect moving from top left to bottom right, suggesting that more data or less sensitive matching can mitigate the effect of unusual training data.

Table 5.8 shows the marginal increase, with ordered as a baseline, of each matching technique for each MDP size, to illustrate the tradeoffs between additional data and matching technique. When considering matching functions, the easiest technical change is from ordered to ordered-1, where one premise is removed from the test state before comparison with the MDP states. In all cases, the probability of providing these hints is higher than that of providing hints based on unordered matches. This is probably because there is some inherent partial ordering in proofs, so only limited benefit is seen from reordering premises. When an ordered hint cannot be matched, it is perhaps more likely that the student has just performed a step that no one else has done before, rather than generating a new ordering of steps, so the benefit of ordered-1 can exceed that of unordered. Providing the unordered search requires us to maintain 2 separate MDPs to make the search more efficient, so there are both time and space tradeoffs to using unordered matching. However, adding unordered-1 after adding unordered provides a very large difference in our capability to provide hints, with little investment in time. As part of this study we also compared the unique states across semesters, as shown in Table 5.9. This gives us a measure of the percent overlap between MDPs. Using 3 semesters of data with ordered matching, or using 1 semester of data with unordered-1 matching, both give us over 50% matching of states across MDPs. When compared with the much higher move matches, this suggests that although a new semester may bring many more different solution steps, the ones actually used for complete solutions already exist and are those most often used by students.

Table 5.8. Marginal increases when comparing matching techniques to ordered

Technique	1-sem. ordered	2-sem. ordered	3-sem. ordered
Unordered	6.83%	5.65%	4.94%
Ordered-1	7.09%	8.27%	9.25%
Unordered-1	12.21%	11.93%	11.64%

Table 5.9. Unique state % matches across semesters and techniques

Test set	1-sem. MDPs	2-sem. MDPs	3-sem. MDPs
Ordered	34.55%	45.84%	51.93%
Unordered	43.62%	55.23%	59.90%
Ordered-1	48.25%	63.07%	71.39%
Unordered-1	57.28%	71.98%	77.87%

5.2.2 Exploring the "cold start" problem

One critique of using data to generate hints has been the expected time needed for the method to be applied to a new problem, or in other words, the "cold start" issue. Our hypothesis was that a relatively low number of attempts would be needed to build an MDP that could provide hints to a majority of students. One method for building our hint MDP would be to incrementally add MDP states as students solve proofs. This experiment explores how quickly such an MDP is able to provide hints to new students, or in other words, how long it takes to solve the cold start problem. For one trial, the method is given in Table 5.10.

Table 5.10: Method for one trial of the cold-start simulation.

1. Let Test = {all 523 student attempts}
2. Randomly choose and remove the next attempt a from the Test set.
3. Add a's states and recalculate the MDP.
4. Randomly choose and remove the next attempt b from the Test set.
5. Compute the number of matches between b and MDP.
6. If Test is non-empty, then let a:=b and go to step 3. Otherwise, stop.

For this experiment, we used the ordered and unordered matching functions, and plotted the resulting average matches over 100,000 trials, as plotted in Figure 5.1. These graphs show a very quick rise in ability to provide hints to students, that can be fit using power functions, whether using ordered or unordered MDP states and matching. Clearly, the availability to give hints ramps up very quickly. Table 5.11 lists the number of attempts needed in the MDP versus target hint percentages. For the unordered matching function, the 50% threshold is reached at just 8 student attempts and the 75% threshold at 49 attempts. For ordered matching, 50% occurs on attempt 11 and 75% on attempt 88. These data are encouraging, suggesting that instructors using our MDP hint generator could seed the data to jump-start new problems. Based on these results, we hypothesized that, by allowing the instructor to enter as few as 8 to 11 example solutions to a problem, the method might already be capable of automatically generating hints for 50% of student moves. This was investigated in Chapter 9.

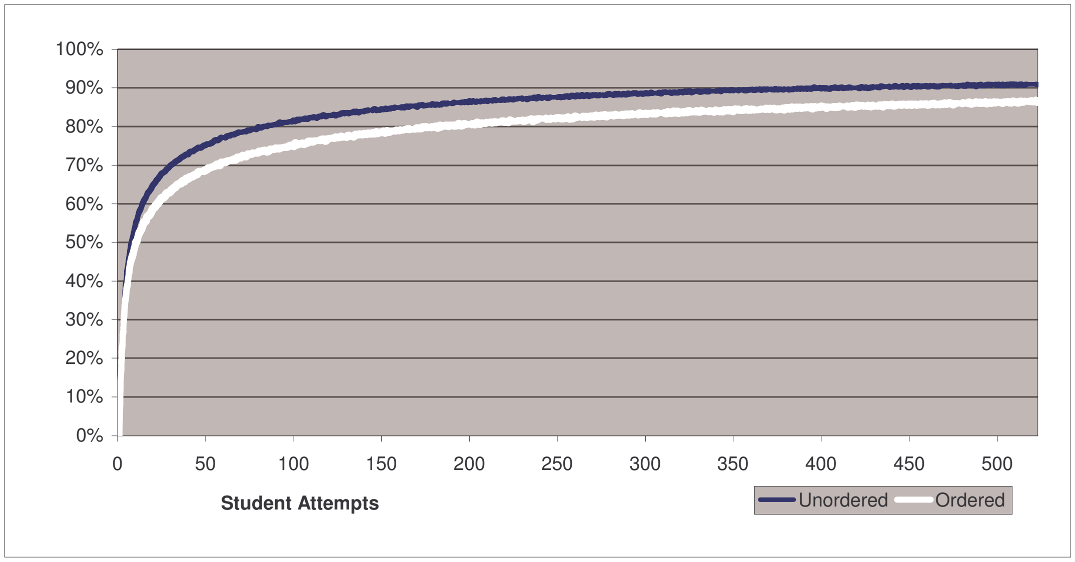


Figure 5.1. Percent hints available as attempts are added to the MDP, over 100,000 trials

Table 5.11. Number of attempts needed to achieve threshold % hints levels

	50%	55%	60%	65%	70%	75%	80%	85%	90%
Un-Ordered	8	11	14	20	30	46	80	154	360
Ordered	11	15	22	33	55	85	162	362	?

5.3 Summary

In this chapter, we have proposed and explored the feasibility of an approach to creating Markov decision processes from student work to automatically generate hints. We show that the method is able to give hints for an average of over 72% of student moves using just one semester of student data. Further our experiments clearly show the method quickly ramps up, but after several semesters of data are used more data offers a limited increase in the percentage of moves covered, indicating valuable tradeoffs between hint specificity and the amount of data used to create an MDP. This approach differs from prior work in authoring tutoring systems by mining actual student data, rather than relying on teachers to add examples the system can learn from. In addition, the generated hints are not created by hand as in example-based tutors, but created based on past student work. Our novel MDP-based approach enables us to automatically provide highly contextual hints, and also allow our knowledge model to learn from new student data. We note that on cold start for a new problem that has no student data, the system will still act as a problem-solving environment, but after even one semester of data is collected, a significant amount of hints can be generated.

CHAPTER 6: THE HINT FACTORY – PILOT STUDY

After developing and validating our methods, the next step was to implement the MDPs into an actual tutor. In this chapter we describe a pilot study to investigate how students use hints, and show that hints help students solve more problems than past classes using the tutor without hints.

6.1 Background

The Hint Factory consists of the MDP generator and the hint provider. The MDP generator is an offline process, but the hint provider must be integrated with the CAI. In this experiment we modify the deductive logic Deep Thought tutor to provide hints while students work problems. The modifications are minimal, including creating a hint generator template, tracking the student actions into a state, passing them to the hint provider, and adding a hint button. All of these modifications need only be done once for a particular CAI.

6.1.1 The MDP Generator

The MDP Generator uses historical student data to generate a Markov Decision Process (MDP) that represents a student model, containing all previously seen problem states and student actions. This approach was outlined previously in Chapter 3. Our method takes the current premises and the conclusion as the state, and the student's input as the action. Therefore, each proof attempt can be seen as a graph with a sequence of states (each describing the solution up to the current point), connected by actions.

Specifically, a state is represented by the list of premises generated in the student attempt, and actions are the axioms (rules) used at each step.

6.1.2 Deep Thought

Deep Thought is a custom CAI tool that allows students to practice solving logic proofs (Croy 1999). As shown in Figure 3.1, Deep Thought's graphical interface allows the students to visually connect premises and apply logic rules. Our new hint button appears, as shown at the lower right in Figure 1, when a student loads a problem that has available hints. The button is bright yellow to make it more visible. When a new problem with hints is selected, the hint provider loads the entire hint file into memory.

6.1.3 The Hint Provider

When the hint button is pressed, the hint provider searches for the current state in the MDP and checks that a successor state exists. If it does, the successor state with the highest value is used to generate a hint sequence. When attaching the hint provider to an existing CAI, we work with instructors to determine the wording and order of hints. However, these variables are easily changed, and experiments can verify the appropriateness and effectiveness of the generated hints.

We worked jointly with two logic instructors to construct an appropriate sequence of hints to generate from successor states. Our choices were based on one-on-one tutoring strategies, research on hint strategies, and consistency with existing tutors. In one-on-one tutoring, both instructors prefer hints that help students set intermediate goals, as has been shown to be effective in (McKendree 1990). Existing tutors use several additional types of hints, including pointing hints and bottom-out hints. Pointing hints help focus user attention, while bottom-out hints essentially tell students the answer

(VanLehn 2006). In this experiment, a hint sequence refers to hints that are all derived based on the same current state. Our logic proofs hint sequence consists of four types of hints: 1) indicate a goal expression to derive, 2) indicate the rule to apply next, 3) indicate the premises where the rule can be used, and 4) a bottom-out hint combining 1-3. For the problem state seen in Chapter 3, Figure 3.1, the hint sequence seen in Table 3.1 would be generated. For each state, four distinct hints are generated. If a student requests a hint, then makes an error, and requests a hint again, the next hint generated is the next one in the current sequence. Once a student performs a correct step, the hint sequence is reset. Whenever the hint button is pressed the hint provider records the time, the hint sequence number and text, and the total number of hints requested so far in the problem.

6.2 Experiment

The main goal of this experiment was to test the capability of the Hint Factory to generate hints for actual students. Our secondary goal was exploratory, to determine how students used the provided hints, to inform our future work. Once the Hint Factory was added to Deep Thought, we generated MDPs and hint files for several Deep Thought problems. Since the current semester was already underway, we chose four level 3 problems from Deep Thought, 3.2, 3.5, 3.6, and 3.8. In case of unexpected errors, we enabled the instructor to quickly disable hints in Deep Thought. Fortunately, the software worked well and this was not necessary. Table 6.1 shows the problems used and the minimum number of rules needed to solve them (as determined by logic instructors).

MDPs were generated using data extracted from Deep Thought solutions from two 2007 Deductive Logic courses (PHIL 2105) taught in the philosophy department: spring (30 students) and summer (20 students). The data were cleaned by removing all incomplete proofs and log files with missing data. Table 6.2 shows the number of student attempts used to create the MDPs for each problem, the average length of the attempt with minimum and maximum lengths. Based on these and the expert data in Table 6.1, the problems can be listed in order of difficulty from 3.6, 3.8, 3.2, 3.5.

Table 6.1: Deep Thought problems where hints were added (> is implies)

Prob.	Problem Description	Expert length	Rules used; features
3.6	-(T&L), -T>-N, -(EvT)/-N	3 steps	DEM,SIMP, MT
3.8	Y=P, -Y>-C, -P=-C /Y>C	6 steps	EQUIV(2), SIMP(2), TRANS, HS
3.2	(A>-B)vC, -C, DvB/-D>-A	6 steps	DS, TRANS, DN, DN, IMPL, HS
3.5	K>M, Z>R, -(K>R)/M&-Z	8 steps	IMPL, DEM, DN, SIMP, SIMP, MP, MT, CONJ

Table 6.2: Characteristics of spring and summer 2007 data used to create MDPs. Attempts include only completed proofs. Length includes correct and incorrect steps.

Problem	**3.6**	**3.8**	**3.2**	**3.5**
# Complete Proofs	26	25	16	22
Average Length	8.0	11.9	11.3	18.8
Std Dev Length	5.9	3.1	4.0	12.4
Avg. Correct Steps	5.5	11.2	9.0	16.7
Average Errors	1.1	1.1	0.9	3.1
Time	3:23	6:14	4:25	9:58

Forty students in the spring 2008 course were assigned to work these four problems (as many times as desired). We hypothesized that, with hints, a higher percentage of students would complete the given proofs. This can be measured in two ways: by class and by attempts. Class participation and completion rates for the experimental class were much higher than the source class. For 2008, the attempt and completion rates were 88 and 83%, respectively, out of 40 students. For 2007, these rates were at most 48%, out of 50 students. This may be due to a novelty effect, since the 2008 class was asked to test hints.

6.3 Results and Discussion

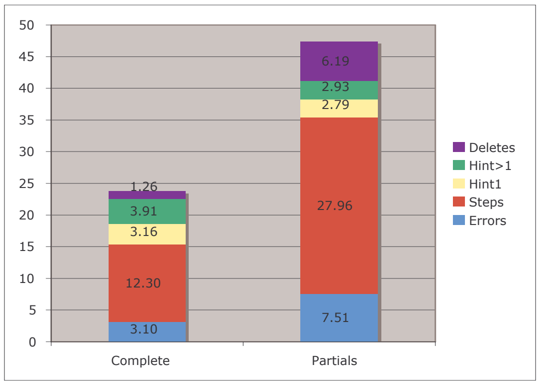


Figure 6.1: Comparison of behavior between complete and partial solutions

Figure 6.2 shows the percent of solution attempts that are complete for the source (2007) and hints (2008) groups. For all problems but 3.5, there was a slightly higher percent complete with hints available. Problem 3.5 showed much higher completion rates for the hints group. Figure 6.1 shows a comparison in behavior during complete and partial solutions. Partial problems were longer by both time and the number of actions. Complete solutions have fewer errors and deletions and more hint usage. These results suggest that some scaffolding may help identify unhelpful behaviors and be used to train students to more effectively learn and use help (Aleven et al, 2004).

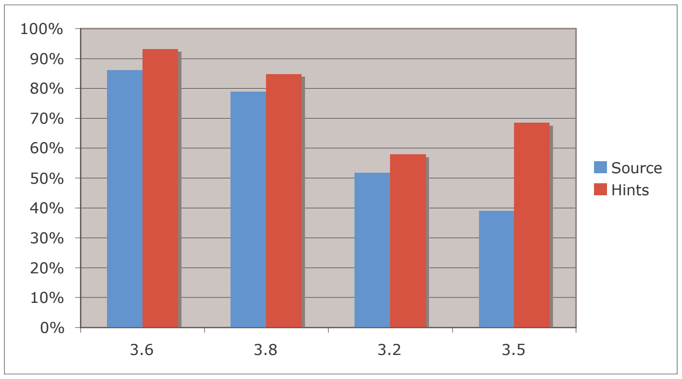


Figure 6.2: Percent attempt completion between the source and hints groups

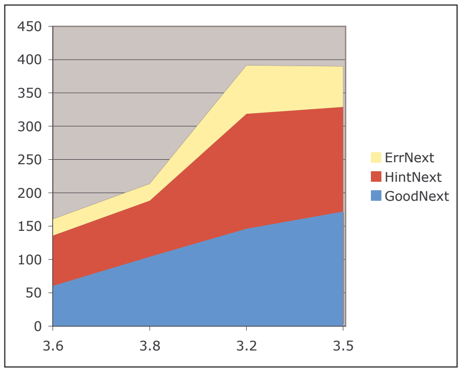


Figure 6.3 Number of steps after a hint that are correct (good) steps, hints, or errors

Figure 6.3 shows the number of hints for each problem, broken down by color into the distribution of hint sequence length. We hypothesized that, as learning occurs, the usage of shorter hint sequences should increase. We do not have reliable data on the sequence of problems that students performed, so we have ordered problems by difficulty. We see here that students did use more hints as we move from less to more difficult problems, and the number and proportion of hint sequences of length 1 seems to go up from 3.6 to 3.8 and from 3.2 to 3.5. Another way to measure the effectiveness of hints is to examine behavior just after receipt of a hint. We therefore investigated the number of errors, correct or good steps, and hint requests immediately following a hint, as shown in Figure 6.3. The proportion of good steps just after a hint goes consistently up, while there is a jump in the number hints and errors requested between problems 3.8 and 3.2. When we examine the difference between 3.2 and 3.5, we see more good steps, and slightly more hints and fewer errors just after a hint. Along with its higher completion rate, this suggests that the hints may be more effective for 3.5.

Table 6.3 shows the hint usage and availability for all 2008 completed and partial attempts. "Moves" is the total number of non-error student actions in the interface. In our prior feasibility study, we built a model to predict the probability that we could provide a hint based on the size of the MDP. In that study, we predicted that proof MDPs built using 16-26 attempts on problem 3.5 have a probability of providing hints 56-62% of the time (Chapter 5). In our current experiment, if a student had pressed the hint button after every move taken, a hint would have been available about 48% of the time. This is lower than our prediction. This difference in the prediction ay be due to student differences in the classes or that the existence of the hint button may have changed student behavior.

Table 6.3: Hint usage and availability by problem, including all solution attempts in Spring 2008

Problem	3.2	3.5	3.6	3.8	Total
Attempts	69	57	44	46	216
Moves	999	885	449	552	2885
Moves w/ Avail. Hints	442	405	230	269	1346
% Moves w/ Avail. Hints	**44.2%**	**45.8%**	**51.2%**	**48.7%**	**47.9%**
Hint1 Requests	236	232	70	154	692
Hint1 Delivered	213	212	66	142	633
% Hint1s Delivered	**90.3%**	**91.4%**	**94.3%**	**92.2%**	**91.5%**

We were encouraged by the comparison of this rate with the hint availability when students requested hints. In Table 6.3, Hint1 Requests counts the number of times a first hint was requested (since hints beyond the first in a sequence are all available if the first one is). Hint1 Delivered shows the number of times a first hint was provided. "% Hint1s Delivered" shows that, over 91% of the times a hint was requested, a hint was available. This percentage quite exceeded our expectations. This suggests that hints are needed precisely where we have data in our MDPs from previous semesters. It is possible that the new paths that were created where hints were not available represent exploratory strategies where students have already determined the apporoach that they are going to take, and therefore they may have high confidence in these steps and need no help.

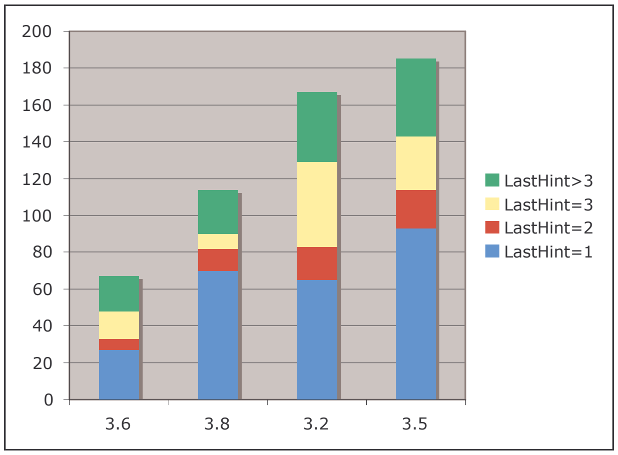


Figure 6.4: Distribution of hint sequences by sequence length.

6.4 Summary

Our tutor can already classify many errors students make, but adding the MDP to this tutor enables it to provide hints. This MDP can constantly learn from new student data. This chapter represents the implementation and a pilot study for our hint generation method in an actual classroom setting. We achieved our main goals, which were to verify that the software would work in a class setting, students would request hints, and hints would be available when requested.

CHAPTER 7: EXPERIMENTAL EVALUATION OF AUTOMATIC HINT GENERATION

7.1 Background

Our spring 2008 pilot study, discussed in Chapter 6, demonstrated that we could successfully use the Hint Factory with the Deep Thought Tutor to automatically deliver context specific hints to students solving four logic proofs. In spring 2009, we added hints to 4 more problems, for 8 total, and examined their use in three deductive logic classes, some with and some without hints. Below are the hypotheses that we tested.

Hypothesis 1: Hints will improve students' ability to solve the proof problems.

Hypothesis 2: System generated hints will help in overall learning of the material. We tested Hypothesis 1 by examining the completion rate of each of the three levels of problems, as well as success on two "no hint" problems given at the end of the third level. We tested Hypothesis 2 by measuring differences between each student's performance on pre and post tests.

7.2 Experiment

We created MDPs using value iteration with the goal state assigned a value of 100, errors a value of -10, and transitions -1, to generate hints for 4 new DT level one problems: 1-1, 1-2, 1-4, and 1-5. We created a "DT-hints" version of DT with hints for these and problems 3-2, 3-5, 3-6, and 3-8, and a "DT-nohints" version without.

We refer to the 3 Spring 2009 classes in this study Online-Hint, Hint, and Non-Hint. The number of students in each class can be seen in Table 7.2. The Online-Hint and Hint classes received unlimited hints on the hint problems and the Non-Hint class received no hints. Online-Hint was an asynchronous online course while Hint and Non-Hint were traditional courses that met on campus. All classes were assigned the problems shown in Table 7.1 (explanation of the symbol notation for these problems is available in Appendix B). In addition to the hint problems there were six additional problems without hints for any classes. Of the six problems without hints, we were especially interested in problems 3-9 and 3-10 since these would represent the last problems the students would have to work, and therefore would provide data to compare learning gains between hints and no-hints classes. Students completed the problems, in order, over the course of the semester. Students accessed the Deep Thought tutor via the learning management system used to administer the course. All three classes used the same system and had the same problems.

Table 7.1. Descriptions for DT problems assigned to the Spring 2009 classes (see Appendix B for symbol descriptions)

Problem	Givens	Conclusion	Hints
1-1	A>(B&C), AvD, ~D&E	B	Yes
1-2	(FvG)>H, IvF, ~I&J	H	Yes
1-3	(~KvL)>(M&N),K>O,~O	N	No
1-4	(~T&S)>~R, ~T, (~QvP)>S, Q>T	~RvN	Yes
1-5	Z>(~Y>X), Z&~W, Wv(T>S), ~YvT	XvS	Yes
1-6	B>(A>J),D&~(A>~C),Bv(A>~C),J>~C	A>~C	No
2-1	(F&I)>M, ~M&(~V>R), (J>K)v(F&I), K>~V	J>R	No
3-1	V>D, ~~~D	~(VvD)	No
3-2	(A>~B)vC, ~C,DvB	~D>~A	Yes
3-5	K>M, Z>R, ~(K>R)	M&~Z	Yes
3-6	~(T&L), ~T>~N, ~(EvT)	~N	Yes
3-8	Y=P, ~Y>~C, ~P=~C	Y>C	Yes
3-9	SvB, B>D, S>G	DvG	No
3-10	J>H, ~W>(~H&~F), J	W	No

7.3 Results and Discussion

After the semester was completed, we analyzed data from the Deep Thought log files. Table 7.2 shows a summary of the number of students in each class, the number of students who completed at least one problem in each of the three DT levels, the number of students who completed the post test, and the number of students who completed the class. All the students who completed the course attempted level 1 problems, but the Non-Hint class had many more students who did not attempt any level 3 problems or who did not take the post test. This result suggests that having hints keep struggling students engaged enough with the tutor to attempt the level 3 problems.

Table 7.2. Profile of number of students in each deductive logic class with level attempts, and course and posttest completion

Class	Hints	Enrolled	Tried Level 1	Tried Level 2	Tried Level 3	Completed Post test	Completed Course
Online Hint	Hints	46	38	32	35	37	38
Hint	Hints	37	28	22	25	28	28
Non-Hint	No hints	37	28	16	18	21	28

Next, we tested Hypothesis 1, by measuring the average number of completed problems for each of the three DT levels, L1-L3, for the hint group (Online Hint and Hint) and the non-hint group (Non-Hint). Table 7.3 shows the average and standard deviation of the number of problems completed by students in each class, in each DT level and also for the separate problems 3-9 and 2-10 which did not have hints for any students. We used two-tailed t-tests to compare the results of the Hint (Online Hint and Hint) group with the Non-Hint group. There were no significant difference in level 2 performance, but level 2 only included one problem assigned. The Hint group performed significantly better, t(N=44) = 2.85, p = 0.007, on L3. This is expected since L3 includes 4 problems with hints, that students are more likely to be able to complete. The Hint group also performed better on the last two questions of level 3, which had no hints, t(N=92) = 1.9, p = 0.06.

Table 7.3. Average Number & Standard Deviation of Completed Problems for DT Levels L1-L3 and Problems 3-9 & 3-10 for each Spring 2009 class

Class	N	L1 Avg.	L1 SD	L2 Avg.	L2 SD	L3 Avg.	L3 SD	3-9 & 3-10 Avg.	3-9 & 3-10 SD
Online Hint	38	4.13	1.80	1.16	1.51	4.13	2.90	1.05	0.98
Hint	28	4.46	1.50	0.89	1.10	5.30	2.81	1.21	0.96
Non-Hint	28	3.54	2.13	1.04	1.73	2.96	3.35	0.71	0.9

Since the online class may have other factors influencing behavior, we performed a comparison between both on-campus courses. Again, there were no significant differences in level 2 performance. The hint group performed significantly better on L3 overall, t (N=54) = 2.80, p = .007, and on the last two L3 problems, t (N=54) = 2.01, p = .049.

We then looked at the likelihood that a student would complete a level 3 problem. Table 7.4 shows the number of students who attempted and did not attempt level 3 for the hint and non-hint groups. This data only consisted of the traditional classes (online class was removed, although adding the online class only increases the likelihood). Based on this information the 89.28% of the hint group attempted level three while only 64.28% of the non-hint group attempted level three. Additionally, holding all other values constant the likelihood of the hint group attempting level 3 is 4.6 times the likelihood of the non-hint group. Regressing level three attempts and no attempts as a binary value onto hint-group in a binary logistic regression showed hint-group to be a significant predictor, X-square(1) = 5.12, p=.024. This means that students who have hints in level 1 are more likely to attempt level 3.

Table 7.4. Overview of Student attempts on Level 3 (traditional classes only)

Group	Number of Students who Attempt	Number of Students who do not attempt	Percentage that Attempted
Hint	25	3	89.28
Non-Hint	18	10	64.28

We hypothesized (H2) that students with hints would have higher pre- to post-test learning gains. The post test consisted of material from the entire course, including problems in logic. There were five logic proof problems in the post test were screenshots from the Deep Thought tutor where the student needed to fill in the next step. Unfortunately, we were not able to show a significant increase in the post-test results of the hint group vs. non-hint group. We believe one reason for lack of increase was the high attrition rate in the non-hint group. Because of the attrition, these students in the non-hint group did not that complete the post-test and we believe they would have scored lower since these students were also the students who did not attempt level three. The pre and post tests also contained material for the entire course and not just logic proof problems. We had hoped to look at just the post test data for just the proof section, but due to the data collection from the learning management system it was not possible.

Several limitations of our study may have affected our results. The syllabus was the same for all three classes, so the intended teaching objectives and assignments were all identical. However, there can be significant differences between professors, between different classes, and between online and traditional on-campus classes. The pre- and post-tests were not specific enough to help us understand the effects of hints, since they included questions from the entire course. To improve the design of the experiment we are performing a crossover of the hint and non-hint groups using the same professors, but switching which professors are using hints in their classes. This will mitigate the differences between professors. Additionally, we will also isolate the proof questions on the pre and post tests to more fully analyze the effect of hints on proofs as well as the entire class.

7.4 Summary

Our main hypothesis in this study was to show that adding hints to Deep Thought would increase logic proof learning outcomes. Although our tests did not show significant differences in pre- to post-test learning gains in the course overall, our results showed that students who received hints on Deep Thought level 1 (L1) were more likely to attempt problems in level 3. This result is important because students who are able to solve the proofs in level 3 are more likely to complete the course and understand the overall material presented in the logic course. We believe this is because the hints during level 1 allowed students who might otherwise give up to continue through the tutor problems. The post-test results did not show any significant results across the hints and no-hints groups, but 25% of the non-hint group did not complete the post-test. Of these 25% who completed the course but did not complete the post-test, only one student attempted level 3 problems. We believe this result shows that only the higher level students in the non-hint group were able to complete the DT problems successfully since most of the students who had difficulty in DT did not reach level 3 and did not take the post test.

CHAPTER 8: UNSUPERVISED MDP VALUE SELECTION FOR HINTS

8.1 Background

In Chapters 6 and 7, we have shown that we can successfully generate context-specific hints for a logic tutor using a Markov decision process (MDP) built from historical student data. The core element of this work that makes automated hint generation possible is the assignment of relative values or "rewards" to each step in a problem solution, since this allows the tutor to identify the action that will lead to the next state with the highest value. Therefore, the value of the state could also be called the expected utility of the state. While a straightforward implementation of value iteration on a MDP has been successful in generating valid hints, there have been instances where the hint was not directly helpful in solving the problem. Therefore, alternate rewards may be needed to avoid "non-helpful" states, or allow for hints tailored to specific student needs or readiness to learn.

As in a recommender system that makes purchase suggestion based on frequent behaviors, we believe that hints generated based on the frequency of a particular step represent those that the majority of students would understand and be able to apply. Vygotsky's theory of the zone of proximal development states that students are able to learn new things that are closest to what they already know (Vygotsky 1987). Presumably, frequent actions could be those that more students feel fluent using.

Therefore, paths based on typical student behavior may be more helpful than optimal or expert solutions, which may be above a student's current ability to understand.

Based on this idea, and the observation that our MDP method sometimes generates a hint that a typical student would not do, or one that is technically correct but was not necessary to the problem solution, we hypothesized that we may be able to generate hints based on "usefulness" and frequency for a particular step in a student's attempt. As explained in Chapter 3, when we construct our MDP, we connect all correct student attempts to a synthetic goal state, assign this state a high value and errors negative values, and rely on value iteration to assign high values to states that are close to the goal, and lower values to those further away. Using this approach results in high values for expert-like solutions, which are short and have few errors. However, in a few instances our tutor gives hints that suggest a less popular path with additional, unnecessary steps. Close inspection of the MDP showed that the states derived from a single, error-free student's solution could get higher values than a more popular route, but that had a significant number of errors. A metric that more heavily emphasizes frequency can mitigate this issue.

The overall goal of our work is to derive domain-independent ways to add intelligence to tutors. However, our typical MDP approach requires that we can label all data as correct or incorrect. In the logic proofs domain, this is simple but in other domains, especially ill-defined domains, hand grading of all student solutions might be required. For example, it is often difficult to determine if a computer program is complete and correct, but it is possible to extract features that many attempts contain, such as variables or loop structures. It seems reasonable to propose that the more student attempts that contain a particular feature, the more likely it is that this feature is a necessary part of a correct program. To lay the foundation for hint generation in such ill-defined domains, we performed an experiment to verify that we could use an unsupervised utility metric to label and value states in an MDP for logic. We hypothesized that this metric would result in similar hints in the logic domain to those we derive using our MDP method.

In our logic proof data, a subset of statements that students use in problem solutions could be considered necessary for problem completion, and therefore all correct attempts will contain these necessary statements. Therefore, it makes sense that a frequency metric that determines how often a statement is used across a student data set might point to necessary steps.

Research on ill-defined domains such as medical diagnosis, computer programming and legal reasoning has shown that it is difficult to generate feedback in environments where there are many possible ways to solve a problem (Lynch et al 2006). For example, in the domain of computer programming, it can be difficult to determine if a program is complete and correct, but it is possible to extract features that many attempts contain such as variables or loop structures. We could suppose that the more student attempts that contain a particular feature, the more likely this feature is a necessary part of the completed program.

Sequential Pattern Matching (SPM) is a data-mining method used in ill-defined domains to extract frequent actions into plans. SPM has been used in a tutor to teach astronauts to use a robotic arm, where the tutor suggested a plan based on their current location in the problem (Nkambou et al 2008). Like our approach, this method only uses good solutions and takes into account how often different actions occur, but is specific to the robotic arm control domain.

8.2 Method

The utility method that we present here shows a capable method for determining the "goodness" of a state by taking into account the state features contained in the state. In our original MDP method, all paths which solved the problem were directed to a goal state which was given a high reward value. The use of a single goal state works well unless we do not know if the student solved the problem in the attempt. So unlike our original method where the goal state was known, the utility method has no known goal states so all terminal, non-error states are treated as a possible goal.

We derive our utility metric using techniques related to Latent Semantic Indexing (LSI), which are used to search large databases of text documents (Landauer et al 1998). In LSI, terms refer to words, while for logic proofs, we define a term, or feature, as the statement a student derives in a single problem-solving step. Therefore, each attempt is composed of a sequence of statements. As in LSI, we use a term-document matrix, as shown in Table 8.2, to show the occurrence of each statement or term in each student attempt, marking a 1 for terms that occur and 0 that do not occur. We then compute the frequency by summing the columns. We set a percentage frequency threshold such that all state features above the threshold had a good potential of being a part of the solution. Setting this threshold can be done automatically or with the help of a domain expert.

Once a list of frequent statements is determined, we calculate initial utility values for all terminal states (leaves) in the MDP. This replaces our original approach of creating a goal state with a single positive value. Valid terminal states are therefore candidate goal states. The utility value of a terminal state is the sum of the value for each step (or feature) in the student attempt. The value of each step is positive if it was frequent and negative otherwise. Error states receive a high negative start value, and all other states start at zero. After the initial values are set, value iteration is applied until the state values stabilize resulting in a value for every state.

8.3 Experiment

We applied the utility method to our NCSU-Proof1 problem and dataset as described in section 4.2. We compared the results of both our traditional and utility methods to assign values to terminal states for this problem, paying special attention to states that had different best values. For these states, we hypothesized that the hints resulting from the utility method would be at least as good as those using the traditional method.

Table 8.1. Sample states derived from example student attempt in Figure 4.1

State	State Description	Error	Action	Result State
1	a → b, c → d, -(~(a → d)		IM	2
2	a → b, c → d, -(~(a → d), -~a v d	Yes		1
1	a → b, c → d, -(~(a → d)		IM	3
3	a → b, c → d, -(~(a → d), a ^ -~d		S	4
4	a → b, c → d, -(~(a → d), a ^ -~d, a		MP	5
5	a → b, c → d, -(~(a → d), a ^ -~d, a, b	Yes		4
4	a → b, c → d, -(~(a → d), a ^ -~d, a		MP	6
6	a → b, c → d, -(~(a → d), a ^ -~d, a, b		S	7
7	a → b, c → d, -(~(a → d), a ^ -~d, a, b, -~d		MT	8
8	a → b, c → d, -(~(a → d), a ^ -~d, a, b, -~d, -~c		CJ	9
9	a → b, c → d, -(~(a → d), a ^ -~d, a, b, -~d, -~c, b ^ -~c			

8.3.1 Data

An MDP was created from the NCSU-Proof 1 data as using the MDP method as described in section 4.2, resulting in 821 unique states. Table 8.1 shows the states created in our MDP for the student attempt shown in Figure 4.1. In the logic domain, a step in the solution is considered to be a new statement added to the previous state. For example, in state 2, the statement ~a v d is the next "step" in the problem, however, since it is an error detected by the software, this statement is deleted and the problem is returned to state 1.

8.3.2 Utility Process

If our data are labeled, we simply connect all valid solutions to a synthetic goal state. However, when goal states are unknown, we need a way to label or measure correct attempts. The utility metric is one way that assumes that frequent features are important in the problem solution.

Table 8.2. Sample matrix showing the occurrence of elements in student solution attempts.

	Terms								
	a → b*	c → d*	-(~(a → d)*	a ^ -~d	a	b	-~d	-~c	b ^ -~c
Attempt 1	1	1	1	1	1	1	1	1	1
Attempt 2	1	1	1	0	0	1	1	1	1
Attempt 3	1	1	1	1	0	0	1	0	0

From our 523 attempts, we extracted 50 unique statements (including 3 given statements) and calculated their frequencies. A partial sample of the statement frequencies is shown in Table 8.2. Note that only the first three attempts are shown and only the statements appearing in those three attempts. The complete term-document matrix, given in Appendix E, contains all 50 statements on the x-axis and all 523 attempts on the y-axis. To determine statement frequency, we sum each column.

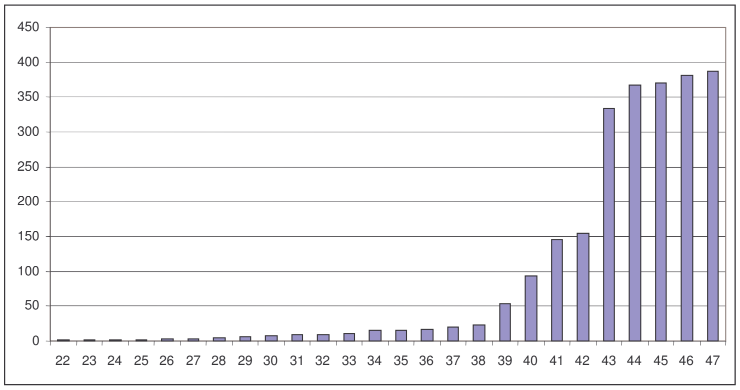


Figure 8.2. Frequency of Statements in Proof 1

To determine a threshold for which steps should be considered "useful," we graphed the frequency of each step (numbered 1 through 47) by their frequency shown in Figure 8.2. Note that the 3 given statements were not included since all attempts had those statements.

Statements 1-22 occurred only once in the data, while statements 43-47 occur in over 370 unique student attempts. Since there is variation in correct solutions, we consider somewhat frequent steps may be useful, so in this case we set a threshold frequency of 8 which is true for statements 29 and higher. We had a logic instructor verify that all the statements at this level could reasonably be expected to occur in student solutions, while those with fewer were not as useful or productive. This threshold value could also be chosen automatically using the frequency profile.

Next we calculate the starting values for states in the MDP. For the possible goal states (terminal, non-error states), the initial value was a sum of the individual scores given to the component statements or “features.” Each statement score was +5 if its frequency was above the threshold and was -1 for those below, and the state value is the sum of its component scores. Error states received a value of -2, and all other states started at zero. Finally, after the initial values were set we applied to the terminal states we ran a value iteration algorithm until the state values stabilized. Note that during value iteration, a -1 transaction cost was associated with each action taken.

8.3.3 Comparing Utility Method to MDP Method

We use an MDP along with its state values to generate hints that provide students with details of the best next state reachable from their current state. To compare the utility method to our traditional MDP method we compared the effects of state values on the choice of the “best” next state. Both methods create the same 821 states, of which 384 were non-error states. From the non-error states, 180 states had more than one action resulting in new state. These 180 states are the ones that we focused on since these are the only states that could lead to different hints between the two methods. Comparing the two methods, they agree on the next best state in 163 states out of 180 (90.56%). For the remaining 17 states where the two methods disagreed, logic instructor experts identified 4 states where the MDP method identified the better choice, 9 states where the utility method identified the better choice, and 4 states that were essentially equivalent. These 17 states can be seen in Table 8.3, with the highlighted cells marking the expert choice.

These results show that the utility method does at least as good a job as the traditional MDP method in determining state values even when it is not known if the student attempt was successful. According to our logic experts, in all cases, the hints that would be delivered with either method would be helpful and appropriate. We believe that the utility metric provides a strong way to bias hint selection toward statements derived by a majority of students, which may give students hints at a more appropriate level.

Table 8.3. States where the methods disagree (17 total states)

State	State Description	# of Possible Actions	MDP next Statement	MDP Value	Utility next Stmt	Utility Value
1	a>b,c>d,-(a>d)	14	-d>-c	49.91	-(-a+d)	10.57
2	a>b,c>d,-(a>d),-(-a+d)	9	b	98.00	(a*-d)	14.00
3	a>b,c>d,-(a>d),-(-a+d),a*-d	8	-(a*-b)	93.00	-(a*-b)	29.00
4	a>b,c>d,-(a>d),-(-a+d),a*-d,b	4	-c	87.72	-d>-c	38.74
7	a>b,c>d,-(a>d),-a+b	6	-d>-c	29.00	-d>-c	18.00
8	a>b,c>d,-(a>d),-a+b,-c+d	2	b+-c	99.00	-(-a+d)	18.02
19	a>b,c>d,-(a>d),-(-d>-a)	2	-(d+-a)	27.13	a*-d	7.67
36	a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d	2	-c	24.33	b	6.04
53	a>b,c>d,-(a>d),-d>-c	5	-(-a+d)	96.00	-b>-a	21.00
82	a>b,c>d,-(a>d),(a*-d),-c	3	b	99.00	-(a*-b)	14.00
91	a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c	3	(a*-d)>(b*-c)	99.00	b	19.33
119	a>b,c>d,-(a>d),-(-a+d),a*-d,b,-d>-c	3	-c+d	98.00	-c	42.71
156	a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b	2	-d>-c	98.00	b	29.60
228	a>b,c>d,-(a>d),a*-d,-d>-c	2	-c	76.20	b	14.00
333	a>b,c>d,-(a>d),a*-d,-c+d	2	-a+b	99.00	-c	19.00
337	a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,b	2	-c+d	61.67	-c	20.20
522	a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c+d,-d>-c	2	-c	99.00	-c+d	30.00

Before we derived the utility metric presented here, we considered modifying MDP values by combining them in a weighted sum with a utility factor after value iteration had been completed. In our first attempt to integrate frequency and usefulness into a single metric, we analyzed all of our attempts to find derived statements that were necessary to complete the proof, by doing a recursive search for reference lines starting from the conclusion back through a student's proof. For each attempt, this "used again" value was set to 1 if a derived statement could be reached backward from the goal, and zero otherwise. We summed the total times a statement was used again, and compared this with the total times a statement occurred in attempts. Table 8.4 shows the comparison of the frequency and used again values for all statements where used again was more than 1. The values have no real correlation, but most items that were used again had high (>7) frequencies, so we decided that frequency was a relatively good indictor of usefulness in the logic proof domain. The "used again" calculation is possible in the logic domain because students must provide a justification for the current statement using rules and references to prior statements. In other domains, this may not be possible but we believe that frequency of occurrence in student solutions indicates that a step is either needed, or is a very common step that will only skew state values in a consistent way.

Table 8.4. Comparison of the frequency and the number of times used again for statements used in proof 1

Statement Number	Statement	Frequency	Used Again
30	(a+c)>(b+d)	8	2
31	-(a*c)+(b*d)	9	2
32	-(d+-a)	9	7
33	(a*-d)>(b*-c)	10	10
34	-(-d>-a)	15	7
35	-b>-a	16	5
36	-(c*-d)	17	6
37	(a*c)>(b*d)	20	4
38	-(a*-b)	23	8
39	(a*-d)	53	44
40	-d>-c	93	71
41	-a+b	145	69
42	-c+d	155	80
43	-(-a+d)	334	300
44	-c	367	344

8.4 Summary

The most important feature of the MDP method is the ability to assign a "value" to the states. This allows the tutor to identify the action that will lead to the next state with the highest value. In this chapter we have shown that, in our logic proof domain, the utility metric for assigning values to terminal states is better than assigning a single goal value to all goal states.

The main contribution of this chapter is to show how this new utility metric can be used to generate MDP values based on features of student solution attempts. Our results show that the utility metric could be used to achieve equivalent or better hints than our prior single-goal MDP approach. This is significant because the utility metric does not require a known goal state, so it can be applied in domains where the correctness of the student attempts is unknown, or difficult or costly to compute. The method of using term-document matrix to determine utility could also be extended into using more complicated Latent Semantic Indexing techniques which would be a natural fit for tutors using textual answers such as essay response questions. Text based answers are prevalent in legal reasoning and medical diagnosis tutors.

CHAPTER 9: EXPERT SEEDING OF MDPS

One goal of our data-driven methods is to be effective quickly. One criticism of data-driven techniques is the amount of time it takes to be able to achieve results for a new problem with no data. Although we addressed this issue with the cold start analysis in Chapter 5, in this chapter we provide an additional method to speed up hint giving capabilities. An expert "seeds" the dataset by completing examples to new problems and we use these examples to create initial MDPs. This process is similar to that used by the example-tracing Cognitive Tutor Authoring Tool, where teachers work example problems that are used as a source to extract problem-specific rules that model how students solve problems. We hypothesize that we can provide significant coverage (over 50%) for hints with just a small amount of time devoted to experts solving a problem by working examples.

In addition to the seeding experiment, this analysis examines additional problems in logic domain order to further validate the work done in chapter 5. We show that three additional problems show similar hint coverage to the problem (NCSU Proof 1) previously studied.

9.1 Background

Historically, the research and development of intelligent tutors have relied on subject area experts to provide the background knowledge to give hints and feedback. Both the cognitive tutors and constraint based tutors that have been the most successful rely on “rules” that the experts create (Mitrovic et al, 2003). This is a time consuming process, and requires the experts to not only understand the subject material, but also to understand the underlying processes used to give help and feedback.

Recently, the makers of the CTAT authoring tools have implemented an “example tracing” tutor (Salden et al, 2008). This type of tutor allows experts to enter in example problems into a tutor and annotate hints as they step through a solution. In many ways this is similar to the seeding approach that we are proposing. There are a number of differences in the approach, the most important is that the expert need not supply the specific hint in our tutor and only needs to provide completed example problems. Additionally, the example tracing tutors will only have the knowledge that the expert has added, while our methods would allow the tutor to continue to improve with additional expert or student problem attempts. Finally, our method still computes a value for the states automatically, and this value allows the tutor to make decisions on which path to suggest even when multiple choices are reasonable. This ability to differentiate between several good solutions based on the specific context of student’s current state remains the strong point of the Hint Factory.

9.2 Method

We performed this experiment using problems 1-4 from the NCSU dataset from fall semesters 2003, 2004, 2005, and 2006 as described in section 4.2. We generated an MDP for each semester of data separately, and Table 9.1 shows the number of states generated and number of total moves generated from each problem during each of the four semesters. Note that problem 1 was used in the original validation experiments that were reviewed in Chapter 5. In Table 9.1, note that the total number of attempts, states, and moves are lower in the Fall 2005 and significantly lower in Fall 2006 semesters. We note that problem 4 has significantly fewer attempts than other problems in every semester, and according to the instructor this is because the problem is more difficult than the others.

Table 9.1. Semester data, including attempts, moves, and states in the MDP for each semester

Problem	Semester	# Attempts	MDP states	# Moves
1	f3	172	206	711
1	f4	154	210	622
1	f5	123	94	500
1	f6	74	133	304
2	f3	138	162	628
2	f4	142	237	752
2	f5	105	122	503
2	f6	63	103	279
3	f3	139	145	648
3	f4	145	184	679
3	f5	113	103	577
3	f6	71	94	372
4	f3	103	46	166
4	f4	59	63	103
4	f5	34	30	48
4	f6	33	20	41

We analyzed these problems using the class cross-validation as in Chapter 5 to see how much coverage the expert seeding provided. Table 9.2 shows the average move matches for each problem when one, two, and three semesters of data are used. The results are similar to those in Chapter 5. With one semester of data the MDP covers an average of 71.46% of all the valid moves seen in a new semester of data. With two semesters of data the average move coverage reaches 77.32% for a 5.86% marginal increase. Adding a third semester of data results in an average move coverage of 79.57%, a 2.25% marginal increase over two semesters. For each individual problem the marginal return on adding another semester decreases as we add each subsequent semester. Problem 4 is interesting in that the total percentage of move matches is approximately 15-20% lower than the other problems. Further investigation of this problem show that the fewer number of attempts to be the main cause of the lower percentages. Our experts note that this problem seems to be more challenging for students.

Table 9.2. Average % move matches across problems using the ordered test sets and MDPs

Problem	**1-sem. MDPs**	**2-sem. MDPs**	**3-sem. MDPs**
1	72.79%	79.57%	82.32%
2	75.08%	80.58%	82.96%
3	79.01%	83.35%	84.89%
4	58.94%	65.77%	68.09%
Average	**71.46%**	**77.32%**	**79.57%**

The seeding data was provided by two subject area experts (Tiffany Barnes and Marvin Croy) who spent less than one hour on each problem making several attempts to solve them. Table 9.3 shows the expert problem attempts used to seed the MDPs for each problem. These problems can be seen in Appendix B. MDP states are unique steps that were seen across all solutions, while the number of moves represents all student steps and counts states multiple times. Between two and four attempts were used to generate problem-specific MDPs with between 8 and 15 total states. When compared to the semester data, these states were clearly “high impact” states since these states included such high coverage of the moves, as seen in Table 9.5, especially for problems 1 and 3. These states included the most used paths to solve the problems by the students.

Table 9.3. Seeding data, including attempts, moves, and states in the MDP for each semester

Problem	# Attempts	MDP states	# Moves
1	3	10	19
2	4	12	27
3	2	15	21
4	3	8	20

Table 9.4 shows the average number of unique state matches that seeding gives compared to MDPs made from one semester of data. Table 9.5 shows this comparison for average move matches, which is more indicative of the percent hint coverage we can provide. Comparing Tables 9.4 and 9.5 shows that although only a small percentage of the seeded states appeared in all of the semesters of class data, they could be used to provide hints for a large percentage of the moves (42.95% on average).

Table 9.4. Average % state matches for seeded MDPs and 1-semester MDPs using ordered matching

Problem	Seeds	1-sem. MDPs
1	6.22%	34.55%
2	11.40%	34.60%
3	7.69%	33.36%
4	12.46%	23.45%
Average	**9.44%**	**31.49%**

Table 9.5. Average % move matches for seeded MDPs and 1-semester MDPs using ordered matching

Problem	Seeds	1-sem. MDPs
1	62.08%	72.79%
2	29.82%	75.08%
3	53.33%	79.01%
4	26.57%	58.94%
Average	**42.95%**	**71.46%**

It is interesting to note the move matches for seeded MDPs vary much more than those for an MDP derived from 1 semester of student data. This should be expected since there are only a few attempts used for seeding. In fact, with further analysis of the individual problems we see for problems 1 and 3 there are two common solutions, and these common solutions were also present in the expert seeds, resulting in high move coverage percent rates (62.08% and 53.33% respectively). On the other hand, problems 2 and 4 have many common solutions (neither problem had a solution that was used in more than 15% of the solutions), which results in much lower, but still promising move coverage considering the small number of expert seed attempts.

9.3 Revisiting the Cold Start

In Chapter 5, our cold start experiment explored how quickly an MDP is able to provide hints to new students, or in other words, how long it takes to solve the cold start problem. In this section, we repeat the cold start experiment for problems 1-4 to compare the results from a cold start to those for problems with an MDP seeded with expert solutions. Table 9.6 lists the method for one cold start experiment trial for one problem.

Table 9.6: Method for one trial of the cold-start simulation.

1. Let Test = {all student attempts}
2. Randomly choose and remove the next attempt a from the Test set.
3. Add a's states and recalculate the MDP.
4. Randomly choose and remove the next attempt b from the Test set.
5. Compute the number of matches between b and MDP.
6. If Test is non-empty, then let a:=b and go to step 3. Otherwise, stop.

In this experiment, we plotted the percent of moves each consecutive student could receive with MDPs derived from both student and expert-seeded MDPs using the ordered matching function. To smooth the curve we took the average move matches over

100,000 trials. The graph of MDP and seeded MDP move matches for problem 3 is given in Figure 9.1. The graph shows how the seeding shifts the initial starting point giving a boost over 50% at the start and then converges back towards the non-seeded curve as the number of attempts increases. By 50 attempts the seeded set is just a few attempts ahead and by 100 attempts the 2 graphs are the same. This shows that over time the seeding loses its effect, but as a boost to the startup of new problems, seeding is very effective. The cold start compared with seeded MDPs is similar for the remaining problems, as shown in Table 9.6. Table 9.7 lists the number of attempts needed in the cold start MDP and seeded MDPs to achieve target hint percentages. Again we see that the seeding of each problem gives an initial boost that fades over time as more student attempts are added, which confirms our hypothesis that seeded MDPs can help provide hints early on when little data is available.

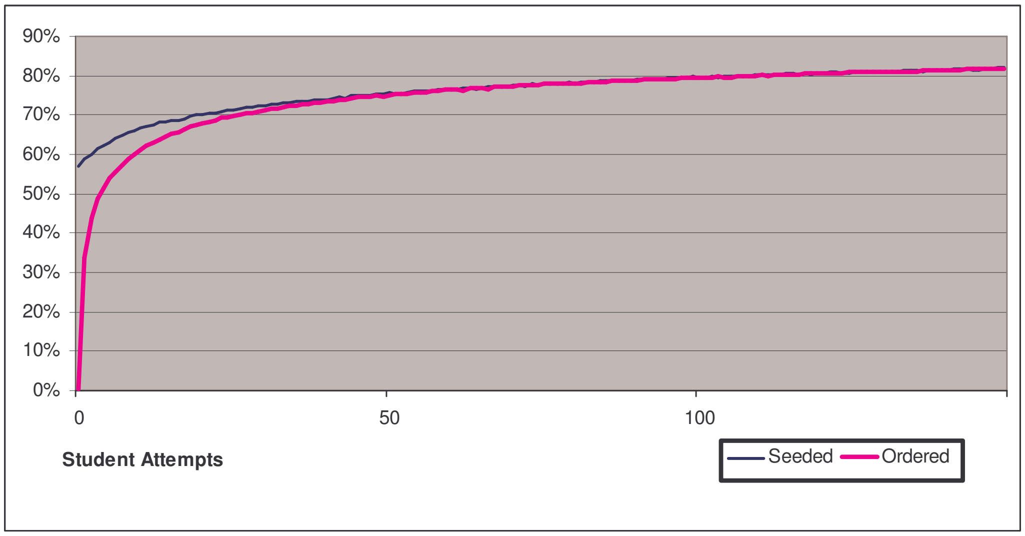


Figure 9.1. Percent hints available as attempts are added to the MDP, over 100,000 trials for Problem 3

Table 9.6. Number of attempts needed to achieve threshold % hints levels

Problem		50%	55%	60%	65%	70%	75%	80%	85%	90%
1	Ordered	8	11	14	20	30	46	80	154	360
1	Seeded/Ordered	0	0	4	8	21	46	80	155	360
2	Ordered	9	15	24	36	59	88	149	286	*
2	Seeded/Ordered	2	3	16	22	46	80	146	286	*
3	Ordered	5	7	10	16	27	50	110	266	*
3	Seeded/Ordered	0	1	3	8	20	48	110	266	*
4	Ordered	25	31	54	82	*	*	*	*	*
4	Seeded/Ordered	12	22	53	80	*	*	*	*	*

9.4 Summary

Although we believe that our method already ramps up quickly as shown in Chapter 5, seeding using expert examples enhances our ability to give hints. For the seeding problems 100% of all the seeded states appeared in each semester of data. Obviously, the educators know the most common solutions to the problems and by seeding the MDPs they can quickly get this data included to help jump start the hint giving process.

Although this experiment showed positive results with historical data, there are some caveats to applying expert seeding to provide hints in practice. The most important caveat here is that hints based on expert solutions may bias students towards trying one of only a few expert solutions. This can result in students not working hard enough to find a solution on their own. It can also limit the variability in the student problem solution space, a characteristic of the type of data we use for constructing effective MDPs that can provide hints for a wide variety of student approaches. Seeding would likely reinforce the expert solution for a long time to come even as additional data is acquired, so the curves shown here for percent move matches may rise much more slowly in practice. To alleviate this problem, instructors can vary which problems get hints each semester so that clean data with no hints can be collected on every problem at some point.

Another potential drawback of using expert seeding is that experts may not include some common or important ways of working a problem in their seeding set. Then, some students would get hints while others with perfectly good solutions might not have hints available. Since we've anecdotally observed that students sometimes modify their proof approaches based on whether hints are available, we believe that the change in hint availability may give the unintentional message to some students that their solutions are not viable or correct and cause them to change course.

CHAPTER 10: USING A BAYESIAN KNOWLEDGE BASE FOR HINT SELECTION ON DOMAIN SPECIFIC PROBLEMS

The major goal of our research has been to provide hints to students with as little human intervention as possible. Our hint generation methods using MDPs have been successful in fulfilling this goal, but are limited to individual problems with previously collected data. To address this issue we have explored a number of additional methods to decrease the ramp up time needed to give hints. One method we explored is "expert seeding" of the MDP (Chapter 9). With experts adding a few problem examples we were able to get an initial set of states that would allow for hints on what the experts thought were good solutions. The drawback of this method is that it does not take into account student preferences for solving the problem and does add an amount of bias towards provided expert solutions. In some domains, logic included, it would be possible to use a proof solver to obtain a solution at any point in the problem. It would be difficult, however, to achieve the context specificity that our method provides by generating hints through exact matches to student attempt sequences, since the proof solver would have to take into account student work in its own solution. We believe that extracting information from multiple problem MDPs and creating a corpus of knowledge that can be used to solve new problems would inherently incorporate student preferences and bias hints toward the rules that students actually use. This may benefit students by providing more easily understood problem solutions and hints.

To accomplish this solution we extracted individual model components (MCs) from each of the MDPs to create a large Bayesian Knowledge Base (BKB) that can then be applied to any problem in the domain. A MC represents a piece of knowledge in the BKB, and in our case this will contain all or part of a student step. MCs are different from Knowledge Components (KCs) as described in Intelligent Tutoring Systems research (Anderson, 1982). A knowledge component in this context represents a concept that a student is learning. Knowledge components are an integral part of knowledge tracing which is used to model student learning and knowledge. Model Components (MCs), on the other hand, are used for model tracing to track what processes students are applying on individual problem steps. A model component could correspond to one or more knowledge components (KCs), and could be tagged with these associated KCs to facilitate knowledge tracing. In the logic domain, some knowledge components represent the ability to apply specific logic rules, and our most specific extracted model components often correspond to individual rule applications.

In the remainder of this chapter, we give an overview of Bayesian Knowledge Bases, explain how our data can be structured in a BKB, and show the results of experiments to test the capabilities of our BKB.

10.1 Background

The main hypothesis of this chapter is that we will be able to combine MDPs on problems with data to create new knowledge in a way that can be used to provide hints for problems with no prior data. Therefore, we need ways to analyze our MDPs and prior data in ways that allow us to aggregate knowledge across these structures. Techniques built on Latent Semantic Analysis (LSA) (Landauer et al 1998) can be applied to domains such as algebra, geometry, and logic to extract individual features of problems and solutions in those domains. In our utility work (see chapter 8), we used term-document matrices as part of the formula for calculating state-action utility.

The use of more complex MDPs such as Hidden Markov Models (HMMs) and Partially Observable Markov Models (POMDPs) have also shown good results in natural language processing (Rabiner 1989) Similar to our MDP method, these techniques apply values to various speaking states in order to determine probabilistically what was said. An HMM can be described as a MDP where some state information is unknown or hidden. In our case, we create our MDPs at a point in time where we assume that the past represents entire possible space even though we know this is not true. It would be possible to use a HMM to include hidden states to model states that we may see in the future but have no information at the current time.

Research into more advanced Bayesian Networks such as Dynamic Bayes Nets (Murphy 2002) or probabilistic Bayesian Knowledge Bases (Antal and Millinghoffer 2005) can provide a basis for extracting parts of the MDP allowing the individual state action pairs to be reused and potentially allow for the emergence of production rules. These production rules could be applied to any problem, including new problems where no previous data had been collected. For this research, we chose to use the Bayesian Knowledge Base (BKB) method to expand our individual MDPs into a full corpus of domain knowledge, since the breakdown of the MDP states into if-then steps closely resembled the individual nodes in a BKB.

Bayesian networks (BN) have been extensively used in intelligent tutoring systems for knowledge tracing, to model the probability of knowing a certain skill (Baker et al. 2008) (Arroyo and Woolf, 2005). Bayesian knowledge tracing incorporates probabilities that a student sometimes gets a problem correct when they do not know the skill by guessing, and conversely sometimes gets a problem wrong when they know the skill (called slipping). Bayesian networks have also been used in model tracing in the Andes physics tutor, which predicts which strategy a student is using by using a solver to create all possible solution paths and then applies probabilities to those paths using likely sequences (Conati et al. 1997). This was possible due to the small number of steps required to solve a problem in the Andes tutor. In the logic proof domain it would not be feasible to compute all possible solution paths.

A BKB is a generalization of a BN where each node has independence from the others (Santos and Santos, 1996). This independence is important because, without it, no problem-solving steps could be separated from the others. BKBs have been used with success in areas where complete knowledge is not available and a full BN cannot be constructed such as in natural language processing (Haddawy, 1994). Figure 10.2 shows an example of a partial BN on the left and the resulting BKB records on the right, which are subgraphs of the BN. The BKB can be thought of as a group of small Bayes Nets where each Bayes Net consists of a parent node and its children connected by actions that have probability distributions. By matching nodes and edges in a particular problem state to those in one or more of our BKB components we can use the Bayesian relationships in the best components to suggest a next step. In other words, given a certain state and its matched model components, probabilities can be calculated within each model component to determine the next action-state pair with the highest likelihood of success. In our implementation the BKB is a group of model components where each component consists of a state, actions, and new resulting states, and we compute a BKB value for each resulting state that will be used for hint selection.

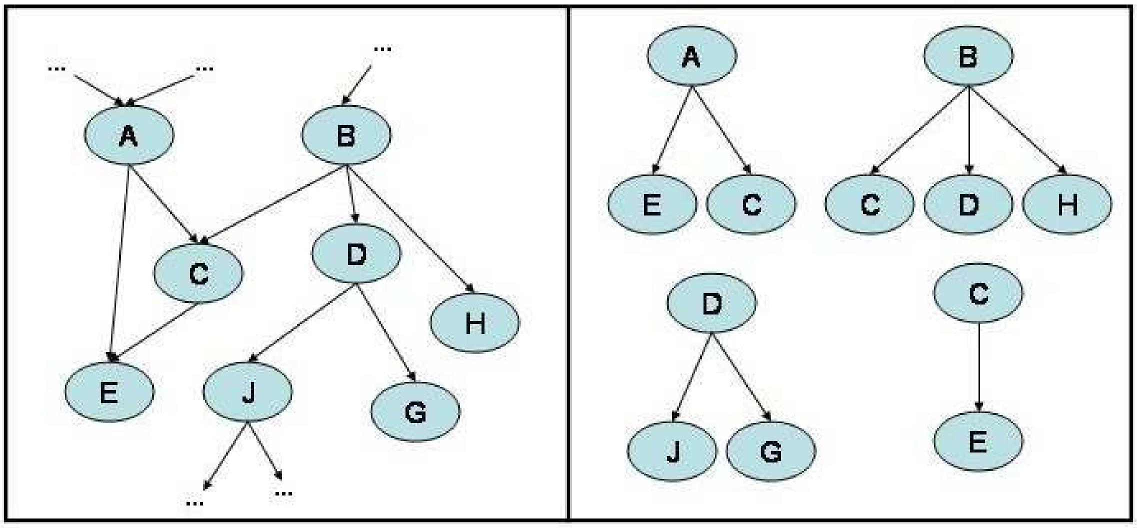


Figure 10.2. Partial Bayesian Network (left) and extracted BKB Model Components (right) (state labels are the same for both)

We know that our data contains certain patterns that we can exploit to create general domain knowledge that could be used to predict new problem solutions and perhaps help in hint generation. The logic domain is a good candidate for study since there are actual rules that we can discover, so we can compare the derived rules with those used in the domain, such as those used in logic proof solvers (e.g. McCune & Shumsky, 2000). However, the best available theorem provers are not easily exploited to provide appropriate help, especially to beginning students, because the proofs they generate do not correspond to the reasoning patterns commonly deployed in "natural deduction" proof systems. Our goal is to use the derived model components to determine the next best state for a student and provide context specific hints that would lead students to that state.

10.2 Method

BKB model components (MCs) can be constructed at three different levels of granularity, as shown in Table 10.1. Level 3 MCs correspond to creating a new MC for each “parent” state in the MDP – consisting of that state, and all the successive action-state pairs that can be taken from that state. Level 2 MCs are level 3 MCs split up so that they contain only one type of action, but can contain multiple new states if the same action is used to derive them. Level 1 MCs contain the smallest knowledge component that can be extracted and consist of the specific part of a state used by an action, the action, and the resulting part of the new state. In practice in the logic domain, a level 1 MC usually represents a logic rule application. Level 3 MCs contain the most context specific information, but will be the hardest to match to student problem states since the MCs describe a specific situation and include multiple premises and logic statements. Before use, all states in the BKB are normalized by replacing problem-specific letters with variables so that they represent more general problem structures.

For the experiments in this chapter, we determine the effectiveness of our BKBs by determining how many states in an unseen problem can be matched with model components in the BKB for hint generation, similar to the cold start experiment. To do this we use only the level 1 MCs since they are the most general and the level 2 and 3 MCs would be redundant in terms of problem space coverage (since level 2 and 3 MCs are composed of level 1 MCs).

Table 10.1. Levels of Model Components and Description

Level	Description
1	The smallest component of knowledge in our BKB. It is just the needed statements, an action, and new statements.
2	Intermediate step, can contain multiple actions.
3	The most robust. It is an entire MDP state and actions

Table 10.2 shows an example state and its successive action-state pairs from Deep Thought problem 3-6 (see Appendix B for more detail on this problem), from solutions students derived in Spring 2009. The MDP Values for the successor states in the table are the same as those in the Spring 2009 MDP. These values are retained as each MC is broken down into smaller parts. This table also represents a single level 3 MC: a normalized copy of the state from the original MDP. This MC is then broken down into 3 level 2 MCs, which can be represented by separating the table into three parts, one for each type of action from state 2 to another state. The level 2 MCs are further broken down to the four level 1 MCs shown in Table 10.3. Level 2 and 3 MCs encode full sets of states, actions, and new states, including all features of the individual problem, even those that are not relevant to the current actions. On the other hand, level 1 MCs contain only the state features used for the given action and those new ones resulting from the action. In the logic domain, correct level 1 MCs correspond to application of standard logic rules, while level 2 and level 3 MCs correspond to full problems with partial steps worked.

Note that one of the Level 1 MCs in Table 10.3 has a MDP value of -10, indicating that its action results in an error state. Although it can be useful for analysis to keep errors in the BKB, we remove them when using the BKB for hint generation. We don’t remove them before MDP derivation because the error states are important in the formulation of the MDP values for all the non-error states as well.

Table 10.2. Level 3 Model Component derived from State 2 from the Spring 2009 MDP for Deep Thought problem 3-6

State	Description	Actions	New States	MDP Values
2	~(T&L), ~T>~N, ~(EvT), ~E&~T /~N	F-SIMP	3	84.00
		F-DEM	6	96.33
		F-TRANS	63	-10.00
		F-TRANS	64	68.25

When constructing a BKB from multiple MDPs, we break each MDP into Level 3 MCs, and break these further down until we have level 1 node-action-node triplets, where the node descriptions are those that are relevant to the current action, as shown in Figure 10.3, and combine all of these level 1 MCs into a BKB. There will be a one to one mapping from the level 3 BKB records and the states in the MDP, however the number of level 1 MCs corresponds to the number of rule applications that have been seen in the student data. We compute BKB values for each successor state in an MC from the MDP values, and these BKB values will be used to select the best action for the student to take in hint generation.

Since we’re going to create a BKB from MCs derived from different MDPs, it is important to normalize their parameters so the MCs from one MDP do not dominate the derived BKB because its values are in a different range than another’s. To maintain this needed consistency, the same parameters for the goal, errors, and state transitions were used to generate each MDP. These values were 100, -10, and -1 respectively. We then save each MDP state with its successors (e.g. the state, its actions and resulting new states) as a Level 3 model component. We give each of these MCs in the BKB their state values and transition probabilities from the initial MDP. A level 3 component will only be seen once in an MDP so the values can just be transferred. The initial BKB values and transition probabilities for level 1 and 2 MCs are the average value and transition probabilities of all the Level 3 MCs that contain them.

Table 10.3. Level 1 Model Components derived from State 2, Problem 3-6 of DT dataset.

State Features	Action	New State Features	Value
~A&~B	F-SIMP	~B	84.00
~(A&B)	F-DEM	~Av~B	96.33
~A>~B	F-TRANS	~B>~A	-10.00
~A>~B	F-TRANS	B>A	68.25

We combine two BKBs by taking the union of all of their MCs, and modifying their BKB values using a weighted sum, discounting the MCs in one BKB by an alpha between 0 and 1. For our experiment we chose alpha=0.5 for the older problem in the BKB (but this parameter can be empirically tuned for individual problems or domains). We then recalculate the transition probablilities based on the combined MCs. If the problems in a set are ordered from easier to harder, as many problem sets are, this means that MCs used in later and therefore harder problems will have higher weights in the BKB. This makes it so that using a BKB for a harder problem will be skewed toward giving hints related to MCs used in the most recent problems. If this is not appropriate for a problem set, the alpha values can be modified for appropriate weightings, either through consulting with domain experts, or by empirical evaluation (e.g. using different alpha values and testing the hint coverage and appropriateness for the resulting BKB).

10.3 Experiment

We hypothesized that we can derive meaningful MCs from problems using the BKB method. The features in a MC can be matched to the features in a students current state and the transition probabilities and values allow us to determine the next best state the student can reach. We can then generate a hint that will take the student towards this state. To test this we create a BKB from problems in our logic data sets and verify that the level 1 MCs correspond to valid rules in the logic domain, and can be used to generate hints for different problems. Also, different problems will have different MCs that are of varying importance to each problem. This will be shown by the distribution of MCs across different problems. This means that some problems will favor BKB items higher than others, but the distribution will be smoother as a variety of different problems are added. Even though these are different, they can be combined to make a general rule set that can be used to solve a new proof problem. We show this by deriving a BKB from a set of problems and testing the MCs on a new problem to see if we could select rules in such a way that the problem can be solved by applying the hints it suggests in succession. In domains that have clear and definable rule sets, the BKB can be used to extract these rules from individual problems and generalize them to be used on new problems. Most mathematics domains would be candidates for this method. In fact, any domain that could be implemented with a production rule system should also work with this method, since the effect of breaking out the individual MCs is to create components that are similar to production rules. In the logic domain this will show how well the rule set gets covered. The experiment is a leave-one-out cross validation where all but one problem is used to make the BKB and then the remaining problem is the test case for the BKB.

We used MDPs derived from the NCSU dataset (1,2,3,4) and the DT dataset (1-1,1-2,1-4,1-5, 3-2, 3-5,3-6,3-8). Problem descriptions for these are given in Table 10.4 and information on the notation is given in Appendix B. We note that problems 1-4 in the NCSU dataset are similar in difficulty to level 3 problems in Deep Thought.

From these data we transformed the Deep Thought data into a format compatible with the NCSU data. We note that although both of these data represent solving logic problems there are differences in the way the two CAIs work. Deep Thought has a graphical interface and allows students to work forwards or backwards to solve proofs (Croy 1999), while the NCSU CAI allows only forwards actions. Deep Thought also allows students to delete steps in their problems. To make sure that the hints would be acceptable for both data sets we eliminated student attempts in the Deep Thought data that used backwards solving and we removed all delete actions. Using the MDP files we extract the individual states as level 3 model components and place them into a Bayesian Knowledge Base (BKB). The steps are further broken down into level 1 model components that are represented by statements, an action applied to the statements, and resulting statements. The level 3 model components represent the specific problem while the level 1 MCs represent the generalized knowledge from the problem.

Table 10.4. Problem Descriptions for NCSU and DT Datasets

Problem	Givens	Conclusion
1	A>B, C>D, ~(A>D)	B&~C
2	A>B, ~C>D, ~Bv~D	A>C
3	(BvA)>C	A>(B>C)
4	Av(B>C), BvC, C>A	A
1-1	A>(B&C), AvD, ~D&E	B
1-2	(FvG)>H, IvF, ~I&J	H
1-4	(~T&S)>~R, ~T, (~QvP)>S, Q>T	~RvN
1-5	Z>(~Y>X), Z&~W, Wv(T>S), ~YvT	XvS
3-2	(A>~B)vC, ~C,DvB	~D>~A
3-5	K>M, Z>R, ~(K>R)	M&~Z
3-6	~(T&L), ~T>~N, ~(EvT)	~N
3-8	Y=P, ~Y>~C, ~P=~C	Y>C

After creating level 1 MCs, we immediately noticed that the NCSU dataset resulted in many more MCs than were derived from the Deep Thought data. Some of the discrepancy was due to invalid data in the NCSU dataset. 16 attempts had invalid items, and the entire attempts containing these were removed. The resulting number of MCs for each of the problems in our BKB (excluding error states) can be seen in Table 10.5 and a list of the MCs is available in Appendix F.

The NCSU dataset still has more MCs, and the remaining discrepancy can be explained partly due to the interfaces and partly due to the specific problems. The NCSU interface is text based while DT is graphical. The graphical interface in DT is more structured allowing students to use the mouse to select rows and create statements. The text-based interface of the NCSU proof tutor is more general, allowing students to enter new statements via the keyboard. This allows students to enter a much wider range of erroneous data when solving the proofs. Also, the DT interface has an overall smaller rule set. Additionally, the rule sets in DT are clearly broken down by inference rules and replacement rules in the interface such that students are unlikely to try rules outside of the ones they are studying for a particular problem. These differences do affect our ability to give hints between the two tutors; this is further discussed in the results section.

Table 10.5. Model Components generated from problems.

Problem	Data Set	Level 3 MCs	Level 1 MCs
1	NCSU	302	58
2	NCSU	428	47
3	NCSU	366	44
4	NCSU	522	78
1-1	DT	248	15
1-2	DT	139	9
1-4	DT	261	12
1-5	DT	342	20
3-2	DT	217	33
3-5	DT	277	40
3-6	DT	246	14
3-8	DT	193	31

Problem 4 in the NCSU dataset has the most states and the most total MCs. According to the instructor, this was the most difficult of the 4 NCSU problems, prompting more exploration in student attempts to solve it. In order to create a Level 1 MC, a rule does not have to be essential to the solution, only it needs to be applied correctly in a problem attempt, and then later the solution needs to be reached. It turns out that many unnecessary rules do get applied correctly in some problem attempts, but then the resulting statements are not used to reach the final solution. Again, this type of behavior was seen much more in the NCSU data compared to DT. This behavior was likely due to the built in scaffolding that DT provides where students can graphically see the connections between the statements. Although the NCSU proof tutor requires students to justify each statement with the line number of a previous statement that was used, it is harder to see the overall structure of the proof solution graph.

After converting the problem MDPs into model components, we compared the coverage of level 1 MCs for each problem. In other words, we use the level 1 MCs from one problem as the source for hints and compute what percentage of the level 1 MCs in the target problem exist in the source. This gives us an idea of the probability that an individual step (corresponding to a single action) will have an available hint using the source problem to derive a BKB, as in the cold start experiment.

Table 10.6 shows this analysis for each problem where source problems are listed in the rows and their level 1 MC percent overlap is given for all the other problems used as targets, given in columns. There was a wider range for the percentage of overlap than we expected. We had expected problems that contained the same rule applications to match at a very high rate; however this was not the case. Further investigation showed that the level 1 MCs for a problem still included specific features of the problem which made the same rule application appear different in another problem. For example, the two level 1 MCs for the application of Simplification in DT problem 1-1 appeared as:

A&B SIMP A
A&B SIMP B

While in DT problem 1-2, the application of Simplification looked like:

~A&B SIMP ~A
~A&B SIMP B

And in NCSU problem 4, the application of Simplification looked like:

(A&B) & (A& C) SIMP (A&B)
(A&B) & (A& C) SIMP (A&C)

Essentially, all of the MCs are equivalent, but our matching function did not match them as the same, because we are simply doing a straight match. This raises the question of whether we should further generalize our MCs. This is certainly possible in the logic domain, but would require additional domain knowledge. The cumulative effect of adding many problems to the BKB, which is described next, mitigates this issue and the detail of our level 1 MCs. When we offer help, this will cause help messages to be more tailored to what students do in a particular problem, and will add more context specificity to the BKB method. This is important because these small differences in rule applications and whether a negation occurs in a statement can make a big difference in students' ability to apply a rule (Newell et al 1957).

Some similar source problems do have much higher match rates as seen in the lower right hand corner of Table 10.6. The level 3 DT problems show a relatively higher match rate with the other level 3 DT problems than with the NCSU Proofs tutorial problems as seen in Table 10.7. Also note that DT problem 1-2 had the overall highest match rate. DT 1-2 had the fewest number of MCs and was the most straightforward problem in the set, and this may mean that its MCs reflect general rules that apply to many problems. On a high level this match rate shows the number of low level features that are present in each of the problems. In the logic domain these level 1 MCs translate into the overlap in the rule set usage among problems.

Table 10.6. Comparison of Match % of problems on Level 1 MCs. (Rows are sources, Columns targets)

Target:	1	2	3	4	1-1	1-2	1-4	1-5	3-2	3-5	3-6	3-8
Source												
1	NA	7.4	4.4	13.4	8.9	8.9	8.9	8.9	5.9	20.9	13.4	4.4
2	8.7	NA	8.7	15.7	10.5	8.7	5.2	17.5	7.0	12.2	7.0	10.5
3	5.7	9.6	NA	13.4	5.7	1.9	0.0	5.7	7.6	7.6	5.7	5.7
4	6.2	6.2	4.9	NA	6.2	4.2	2.8	6.2	4.2	6.9	5.5	4.2
1-1	25.0	25.0	12.5	37.5	NA	33.3	16.6	29.1	25.0	37.5	29.1	16.6
1-2	40.0	33.3	6.6	40.0	53.3	NA	20.0	40.0	20.0	40.0	33.3	20.0
1-4	37.5	18.7	0.0	25.0	25.0	18.7	NA	25.0	6.2	43.7	25.0	6.2
1-5	20.0	33.3	10.0	30.0	23.3	20.0	13.3	NA	26.6	36.6	23.3	30.0
3-2	10.0	10.0	10.0	15.0	15.0	7.5	2.5	20.0	NA	17.5	17.5	15.0
3-5	27.4	13.7	7.8	19.6	17.6	11.7	13.7	21.5	13.7	NA	27.4	19.6
3-6	39.1	17.3	13.0	34.7	30.4	21.7	17.3	30.4	30.4	60.8	NA	39.1
3-8	7.6	15.3	7.6	15.3	10.2	7.6	2.5	23.0	15.3	25.6	23.0	NA

Table 10.7. Comparison of Match % of problems on Level 1 MCs and complete problem sets.

Problem	NCSU	Deep Thought	All Problems
1	8.40	10.03	9.58
2	11.03	9.83	10.15
3	9.57	4.99	6.24
4	5.77	5.03	5.23
1-1	25.00	26.70	26.08
1-2	29.98	32.37	31.50
1-4	20.30	21.40	21.00
1-5	23.33	24.73	24.22
3-2	11.25	13.57	12.73
3-5	17.13	17.89	17.61
3-6	26.03	32.87	30.38
3-8	11.45	15.31	13.91

To provide hints with the BKB method, the current state in the new problem will be broken down into its own MCs and then these will be matched against the BKB. The current state's MCs are matched using a straight, ordered matching function. All matching model components will be returned, and the BKB values will be compared for all action-state pairs, and the one with the highest BKB value will be selected for hint generation, as in original MDP method. To test if the BKB method is able to give hints we use the 11 other problems as source and the current problem as a target, and compute how many MCs in the current problem are matched in the source MCs. If all of the MCs in the current problem are matched, then that means we can generate hints on 100% of the states in the new, unseen problem.

We found MC matches for all problems except DT 3-8. This is the only problem in the set that requires the "Equivalence" rule in order to reach a solution. When we used source BKBs derived only from the same tutor (NCSU or DT), all MCs for each problem were matched, except for problem DT 3-8. This shows that we would be able to give hints using the BKB with most new problems unless they contained rules that had not previously occurred in a problem data set.

We hypothesized that the MDP method of generating hints will be more context-specific and more valuable for students, but that the BKB method will generate some of the same hints, or at least ones of comparable quality. To test the BKB's hint generating ability, we compared the hint-generating states from a BKB to those with an MDP for problem 1-5. In other words, we made a BKB with all the other problems in the set, and compared which model components would be used to generate hints with those in the 1-5 MDP that would be used to generate hints. Out of the 158 states that had more than one choice for hints, the BKB and MDP method agreed on 139 (87.97%). Of the 19 states remaining, 14 suggested the "Delete" action, which was not included in the BKB. The remaining 5 items that disagreed are shown in Table 10.8. In all cases, the instructor indicated that both hints were equally appropriate.

Table 10.8. States where the BKB and MDP methods disagree; in all cases experts agree that either hint is appropriate.

State	State Description	MDP Action	MDP resulting component	MC to apply	MC resulting component
1	Z>(~Y>X), Z&~W, Wv(T>S), ~YvT / XvS	Simplification	~W	A&~B, SIMP, A	Z
13	Z>(~Y>X), Z&~W, Wv(T>S), ~YvT, Z / XvS	Modus Ponens	~Y>X	A&~B, SIMP, ~B	~W
16	Z>(~Y>X), Z&~W, Wv(T>S), ~YvT, Z, ~Y>X / XvS	Double Negation	~~Wv(T>S)	A&~B, SIMP, ~B	~W
22	Z>(~Y>X), Z&~W, Wv(T>S), ~YvT, Z, ~Y>X, ~W, T>S / XvS	Conjunction	(~Y>X)& (T>S)	~AvB, IMPL, A>B	Y>T
48	Z>(~Y>X), Z&~W, Wv(T>S), ~YvT, ~W / XvS	Dysjunctive Syllogism	T>S	A&~B, SIMP, A	Z

10.4 Discussion of Results and Summary

The primary findings of this research suggest that we can generalize the MDP method into a Bayesian Knowledge Base, which contain MCs that can be used to solve new problems. We also showed that a BKB composed of our logic proof problems could be used to provide hints in all but one of the other problems. The structure of the MCs can be stated in an "if-then" format that is very similar to the production rules used in the cognitive tutors. For example, the MC's listed here:

A&B SIMP A
A&B SIMP B

could be be turned into the following production rules:

If you have A&B
and your goal is to get A
then use SIMP

If you have A&B
and your goal is to get B
then use SIMP

This is encouraging since the ability to automatically create some or all production rules for a cognitive tutor would save a tremendous amount of time in cognitive tutor development. This type of BKB could be integrated with an authoring tool such as CTAT, to provide non-expert developers a way to generate feedback without manually creating production rules.

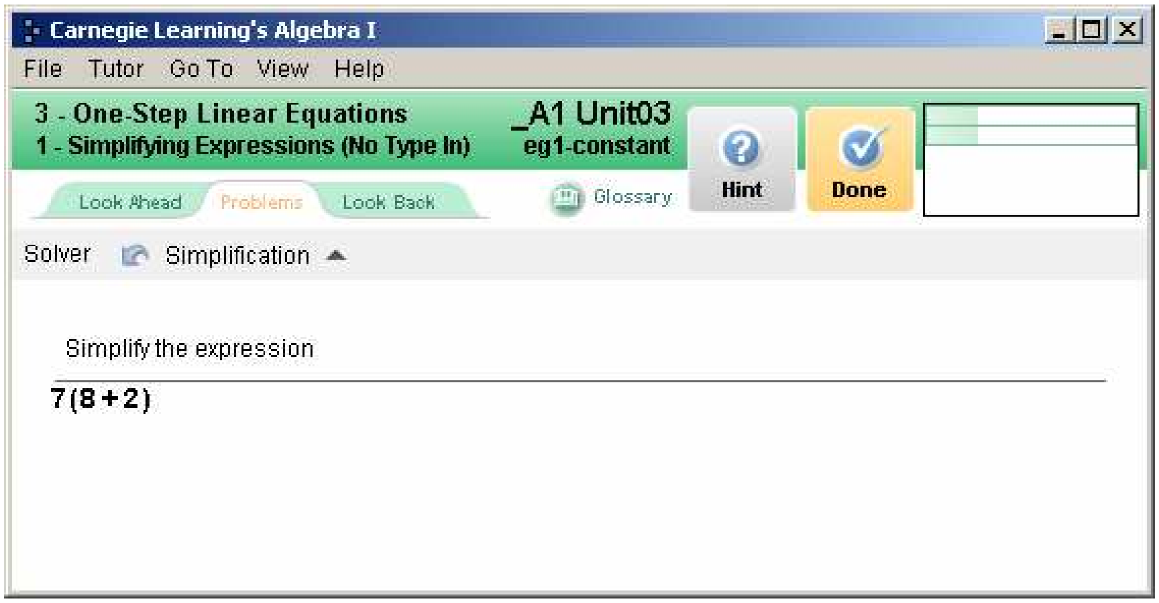
There were some issues that we saw with our MCs. First, we expected the level 1 MCs to generalize down to the specific rule applications. This did occur, but leaving in negations and compound statements caused for more level 1 MCs to be created for many of the problems. This affected our ability to match level 1 MCs across problems, even when the same rules were used. This issue was mitigated when enough problems were used to generate the BKB, allowing for MC matches for almost all the problems in our set. Another issue is that in its current form the BKB will always prefer certain rules much like a production rule system does. In our datasets, the BKB has higher values for Simplification MCs. This is not necessarily bad, but this means that sometimes hints from the BKB would suggest simplifying a statement even if the new statement created was not used again for the solution.

CHAPTER 11: EXTENDING METHOD TO OTHER TUTORS

In this chapter, we show how the MDP method and Hint Factory would work in other domains. We have performed validation experiments on two additional domains. The first is the Algebra tutor from Carnegie Mellon University (Koedinger & Sueker, 1996). This tutor is one of the most studied and refined examples of an ITS, having been in use for almost 15 years. This gives us an excellent opportunity to compare its existing hints to our automatically generated hints. The second is a CAI used to teach a type of Chemistry called Stoichiometry. This CAI currently does not have any adaptive tutoring capabilities.

11.1 Carnegie Mellon Algebra Tutor

Data for the Algebra tutor is publicly available from the Pittsburgh Science of Learning Center's (PSLC) DataShop repository. The DataShop is the largest repository of ITS based data and contains nearly one billion transactions. With over 500,000 students currently using tutors connected to this database the number of transactions is expected to grow at an exponential rate (Koedinger et al 2008). We used the Hampton data set that includes work done in middle school algebra using the tutor during the 2005-2006 school year. An example of the tutor can be seen in Figure 11.1. The data set contains the work of 59 students over an entire school year and includes 8,978 problems with 62,223 unique steps.


Carnegie Learning's Algebra I
File Tutor Go To View Help
3 - One-Step Linear Equations
1 - Simplifying Expressions (No Type In)
_A1 Unit03
eg1-constant
Hint
Done
Look Ahead Problems Look Back
Glossary
Solver Simplification

Simplify the expression

$7(8+2)$

Figure 11.1. Screen Shot of the Algebra Tutor where student is asked to simplify an algebraic equation, the student can enter the simplified expression, ask for a hint via the hint button or complete the problem with the done button.

Upon investigating the data from the algebra tutor, we noted the problems tend to be shorter and more specific to individual skills than in our previous domain (logic). Many of the problems have a single successful path to a solution. While it is possible to apply the MDP method to these problems only one successful attempt would be needed. We do believe that the MDPs generated from these problems could be useful in looking at the possible errors that students make by showing educators how often a particular error occurs at each step. There are a number of problems that are prime candidates for the MDP method. These problems require simplifying equations and have several different paths to the solution. Figure 11.2 shows a problem to which the MDP method was applied. To complete a problem, students select from a dropdown menu what action to perform on the equation (Table 11.1 contains a list of actions that are available in this problem), and then enter the correct numbers for the completed action. Table 11.2 shows three solutions seen for this problem. Solution 1 is the most common solution seen and comprises 70% of the solutions. Solution 2 just changes the order and was seen 12% of the time, while solution 3 is a complex approach that is acceptable by the tutor, but was only seen once.

Solve for x

```
-42-30x-44x = -32
```

Figure 11.2. Example Problem from the Algebra Tutor, Unit 09-EG3.

The MDP computed from the student attempts consisted of 42 non-error states. Of these 42, the most common solution (solution 1 in Table 11.2) accounted for 6 of the states. This solution was the shortest path, and also had the highest values in the MDP.

Table 11.1. Actions available for problem EG3

Actions
Simplification – Add/Subtract Terms
Simplification – Perform Multiplication
Simplification – Simplify Fractions
Simplification – Simplify Signs
Simplification – Distribute
Transformation – Add to both Sides
Transformation – Subtract from both Sides
Transformation – Multiply both Sides
Transformation – Divide both Sides

From the data generated for the MDP, we have devised 2 types of hints per step. The first hint would tell the student the best action to take (e.g. simplification-add/subtract terms), and the second hint would tell the student what they needed to make the equation look like (e.g. -74x = -32+42), which would be the description of the next state in the MDP. In our logic tutor we gave a third type of hint based on what features from the current state could be used to advance to the new state. We cannot offer this type of hint based on the current state features since the features that are used to arrive at the next state are not stored out in the log file for this tutor. The current tutor offers two hints per step as well, but both hints refer specifically to the action to take. The first hint is the more general action (e.g. "Put the equation in its simplest form"), and the second hint tells the more specific action to take (e.g. "Add or subtract terms from both sides").

In the Algebra tutor, some problem states further into the solution, the hints do tell the student which items to apply the action to. Because students do not indicate what parts of an expression they are modifying in a step, the raw data does not reflect this information. It could be derived using domain knowledge about algebra. This is an example where some domain specific knowledge would be needed in order to format the automatically generated hints. The hints from our MDP and the tutor's hints for the initial state can be seen in Table 11.3.

The results of our analysis show that the MDP method is applicable for some problems in a domain like the Algebra tutor. Specifically, these problems have multiple steps and multiple solution paths which allows for the creation of MDPs with values that can be used to direct problem solving. These problems, in general, have less steps and a more structured solution which leads to MDPs with fewer states, but this only increases the effectiveness of the MDP with fewer student attempts. Although these problems have the state-action-next state format that our method uses, the individual problems are not as structured as in the logic domain. This can make it more difficult to provide a general format for hints, and may require annotation or review by educators. However, considering the time that was taken to write out two hints for every step of every problem in the tutor, the MDP method could at least be used to give the educators a list of the possible hints that could be used, and shorten the overall time to annotate solutions with hints.

Table 11.2. Example Solutions from the Algebra Tutor

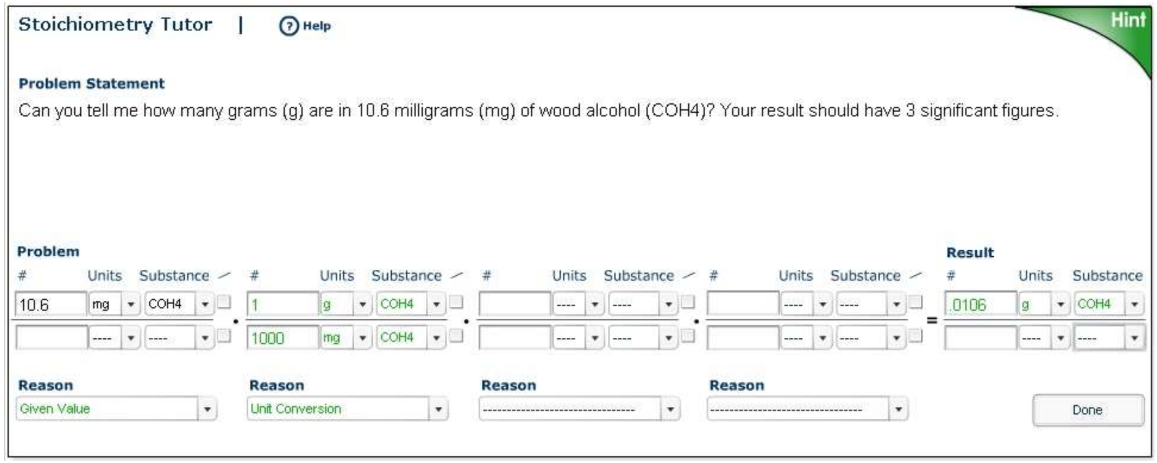
Solution	Description	Action
1	-42-30x-44x = -32	Simplification – Add/Subtract Terms
	-42-74x = -32	Transformation – Add to both Sides
	-42-74x+42 = -32+42	Simplification – Add/Subtract Terms
	-74x = -32+42	Simplification – Add/Subtract Terms
	-74x = 10	Transformation – Divide both Sides
	-74x/-74 = 10/-74	Simplification – Simplify Fractions
	x = -5/37	
2	-42-30x-44x = -32	Transformation – Subtract from both Sides
	-42-30x-44x-(-42) = -32-(-42)	Simplification – Add/Subtract Terms
	-74x = -32-(-42)	Transformation – Divide both Sides
	-74x/-74 = (-32-(-42))/-74	Simplification – Simplify Fractions
	x = (-32-(-42))/-74	Simplification – Simplify Signs
	x = -(-32+42)/74	Simplification – Distribute
	x = (-(-32)-42)/74	Simplification – Simplify Signs
	x = (32-42)/74	Simplification – Add/Subtract Terms
	x = -5/37	
3	-74x+74 = 10+74	Transformation – Divide both Sides
	-74x = 10	Transformation – Divide both Sides
	-42-30x-44x = -32	Simplification – Add/Subtract Terms
	-42-74x = -32	Transformation – Divide both Sides
	(-42-74x)/-74 = -32/-74	Simplification – Simplify Fractions
	-(-42-74x)/74 = -32/-74	Simplification – Simplify Fractions
	-(-42-74x)/74 = 16/37	Simplification – Distribute
	(-(-42)-(-74x))/74 = 16/37	Simplification – Simplify Signs
	-42-30x-44x = -32	Simplification – Add/Subtract Terms
	-42-74x = -32	Transformation – Add to both Sides
	-42-74x+42 = -32+42	Simplification – Add/Subtract Terms
	-74x = -32+42	Simplification – Add/Subtract Terms
	-74x = 10	Transformation – Divide both Sides
	-74x/-74 = -5/37	Simplification – Simplify Fractions
	x = -5/37	

Table 11.3. Example Hints Comparison for State 0 (starting State)

Hint Type	Description
Algebra Tutor – Hint 1	Put the equation in its simplest form
Algebra Tutor – Hint 2	Add/Subtract Terms
MDP Method – Hint 1	Perform Action - Simplification – Add/Subtract Terms
MDP Method – Hint 2	Derive -42 – 74x = -32

11.2 Stoichiometry

The Stoichiometry tutor is a browser based tutor and consists of problems where students have to fill in text boxes and use dropdowns to balance Chemistry equations. A screen shot of the Stoichiometry tutor is given in Figure 11.3, which is available online at: http://learnlab.web.cmu.edu/~pact/chemstudy/learn/tutor1.html. The data for the Stoichiometry tutor was also from the PSLC DataShop, and is available for download with permission. The data set consisted of 1,929 problems, and we analyze problem 1 here. There were 498 student attempts of problem 1 of which 120 were completed and 378 were partial attempts.



Problem Statement

Can you tell me how many grams (g) are in 10.6 milligrams (mg) of wood alcohol (COH4)? Your result should have 3 significant figures.

Problem

#	Units	Substance	#	Units	Substance	#	Units	Substance	#	Units	Substance	#	Units	Substance
10.6	mg	COH4	1	g	COH4		----	----		----	----	.0106	g	COH4
	----	----	1000	mg	COH4		----	----		----	----		----	----

Reason	Reason	Reason	Reason
Given Value	Unit Conversion	--------------------------------	--------------------------------

Figure 11.3. Screen shot of the Stoichiometry tutor where students enter values to complete the equations.

As we discussed in Chapter 3, the most challenging part of implementing the MDP method is deciding how to describe the states and actions. The stoichiometry tutor records a label for each interactive element of the interface, as shown in Figure 11.3, along with what data is selected or entered in that element.. Therefore, it is logical that our MDP states should consist of label-value pairs for each interface element, as shown by the sample state in Table 11.4. Actions in the tutor correspond to a change in values for any element, such as “Numerator1Value.”

Table 11.4. Example State Description for Stoichiometry Problem

State Feature	Value
Numerator1Units	mg
Numerator1Value	10.6
Denominator2Units	mg
Denominator2Value	1000
Numerator2Units	g
Numerator1Value	1

In this tutor, errors are flagged, but remain on the screen until the student changes the value. In the logic tutors, errors never remain as part of a good state that is in the solution path. Errors are terminal nodes and the problem state is returned to its previous state just after the error is made and a message is shown to the student. This simplifies hint generation, since good states never contain errors. We replicate this for stoichiometry. When an error occurs we add the error state, but then revert back to the pre-error state even though the error still exists on the screen.

After the state description and actions were defined we created an MDP for both ordered and unordered states. Ordered states keep the order that the items were filled in as part of the state, while unordered considers all moves with the same items filled in the same state. Table 11.5 shows the number of states generated for the MDPs, which is considerably higher than the logic problems. This difference makes sense since there are 42 separate interactive elements for the students to complete in the stoichiometry tutor.

Table 11.5. MDP States Generated for Problem 1

Type	Total States	Good States	Error States
Ordered	4052	1978	2074
UnOrdered	1798	854	944

Next, we replicated the cold start experiment with this MDP to see how quickly we could generate hints in this domain. Figure 11.4 shows the graph of how quickly the ordered and unordered MDPs can give hints. Table 11.6 shows the number of attempts to reach the percentage thresholds. The resulting curves look very similar to those from the logic data but the ramp up takes significantly more attempts and the highest percentage of hints we can give is at 65% instead of the 95% values we could achieve with enough data in the logic domain. It is possible that, with more data, we could achieve this amount for Stoichiometry, but the curves imply that we may not be able to do better than 70% hints. We believe this has to do with the complexity of the states and the large number of partial attempts.

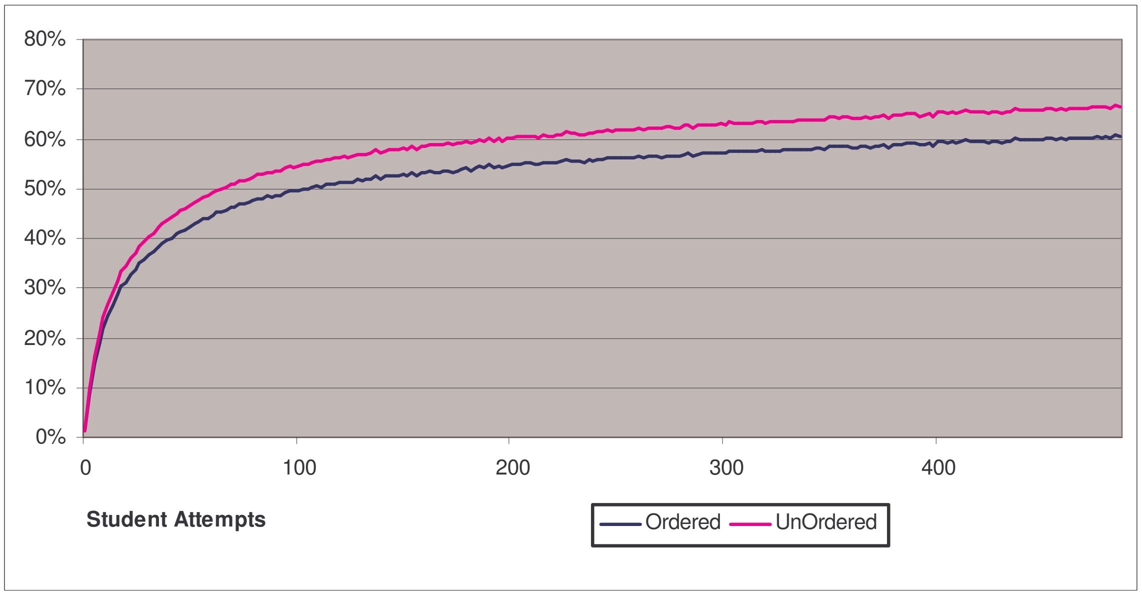


Figure 11.4. Percent hints available as attempts are added to the MDP, over 100,000 trials for Problem 1

Table 11.6. Number of attempts needed to achieve threshold % hints levels

Type	40%	45%	50%	55%	60%	65%
Ordered	52	60	101	189	422	*
UnOrdered	44	51	68	108	192	397

Finally, we looked at the types of hints that could be supplied for this domain. We propose a hint sequence that can be seen in Table 11.7, which would consist of 3 hints. This hint sequence is based on the information that we have about the current state, the actions taken, and the next state. The first tells the student which interactive element to fill in. The second tells the student the correct value to enter. The third hint restates the first two hints together.

Based on the analysis we have performed here, we believe this domain is a good fit for the MDP method. Creating hints for all possible steps in these problems would be extremely difficult for experts to do by hand, since there are so many possible items on each screen. We have offered the MDP method to the developers of this tutor and they are interested in performing experiments to determine the effectiveness of hints in this domain.

Table 11.7 Proposed Hint Sequence for Stoichiometry tutor.

Hint #	Hint
1	State which interactive element to fill in (based on best action)
2	Give the correct value to enter (based on next state)
3	Give the interactive element and value to complete (combination of the above)

11.3 Summary

The primary research goal of this chapter is to show that our methods can be applied to a variety of domains. We have shown two additional and different domains that could benefit from our method. Additional work by Fossati, et al (2009) shows that others have seen the potential uses of MDP for providing adaptive feedback and are starting to apply our methods in their work. Looking at the similarities of the MDP generation process between these very different domains gives support that we can reach our longer term goal of creating a “black box” version of the MDP software that could be easily used to add intelligent hints and feedback to any existing CAI for multi-step problem solving.

CHAPTER 12: CONTRIBUTIONS, IMPACT, AND FUTURE WORK

The main contribution of this work is the automation of the creation of context-specific hints similar to those in intelligent tutors. In the past five years, there has been a trend for colleges to offer more and more instruction online, and now we see this trend continuing to the K-12 level (my kindergarten daughter is using a learning management system and CAI). In, 1993-94, a curriculum designed around the algebra tutor was introduced in the Pittsburgh Public School District and showed impressive results (Koedinger and Anderson 1995). From this success, the cognitive tutors, now distributed through Carnegie Learning, (www.carnegielearning.com), are now being used by more than 500,000 students per year. By automating a portion of the process of creating ITSs from CAI we will be able to take advantage of and promote this trend. The MDP method is the foundation to the proposed "Cascading Hint Factory" which can be seen in Figure 12.1.

The idea of the cascading hint factory is to provide a framework for the automatic generation of hints. Starting with our MDP method, which is the most context specific, the system can try and produce a hint from the student model if it exists. If the current student state does not exist in our model the MDP method is unable to give a hint, so our framework looks to the expert generated "seeded MDP" which has many of the common solutions that experts predict will be seen, but may not support student thought processes as well as the MDP method. If a hint is still not available with the seeded MDP, the framework passes the state on to the BKB method where the features of the current state are matched against the records in a Bayesian knowledge base that was created from similar problems. If appropriate BKB records exist, they can be used to point to the next best action and a hint can be given. If, however, the BKB fails to match any records for a given state, the cascading hint factory can still give a hint using a low level solver. Solvers exist and can be used in a number of domains such as logic (McCune & Shumsky 2000) and Geometry (Bouma et al 1995). Although the solver can provide a solution path that can be used to give a hints, these hints will be the least context specific and least likely to correspond to the way students learn and solve problems.

In the proposed cascading system, one or more of our hint generating methods could be employed to generate hints for any encountered student states. Combined with the ability for teachers to annotate the MDP, this implementation would provide intelligent hints with a much smaller time investment that that needed for cognitive tutors. We envision this cascading hint factory attaching to existing computer aided instruction, collecting student data, building student models, and delivering hints, with little expert human involvement. This is a tremendous contribution to the ITS field: automatic hint generation allows for the quick addition to context-specific hints to existing CAI tools, and the cascading hint factory can also serve as a repository to collect empirical data about learning in the CAI's content domain. This can be used to better understand how students solve problems in the given domain through the implicit domain models being created through the MDP andBKBs.

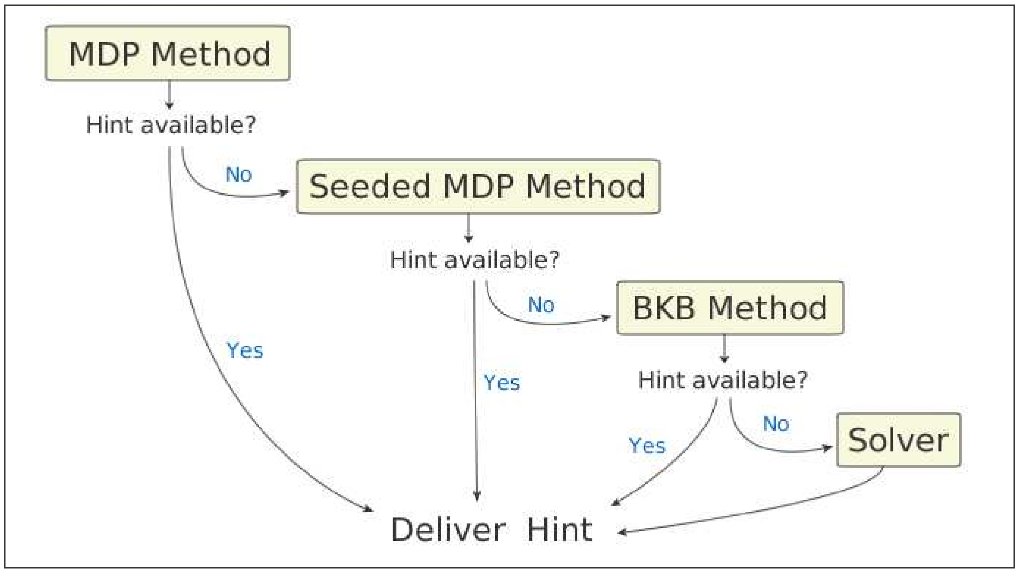


Figure 12.1. Cascading Hint Factory framework for automatically generating hints

12.1 Contributions

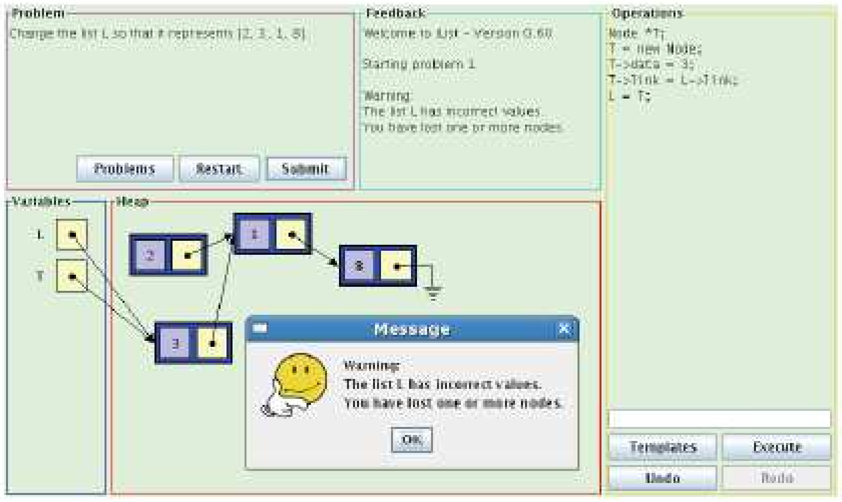
In this dissertation, we have presented a novel method of automatic hint generation using past student data. These hints are highly contextualized for a specific problem state. This method which is implemented using Markov Decision Processes, as presented in chapter 3, forms the foundation of the Hint Factory. Using the hint factory we can automatically generate hints for students in a number of domains. The majority of the research here focused on tutors for solving logic proofs. In chapter 3, we introduced the MDP method for generating hints. We have validated the MDP method for hint generation (chapter 5 and 9) and implemented it in a live classroom (chapter 6 and 7).The MDP method represents the most context specific way to automatically generate hints. We discussed the need for data and showed that in the logic domain the method quickly ramped up with a minimal number of student attempts (chapter 5 and 9). However, to improve the hint giving capabilities, we introduced the addition of a utility metric (chapter 8). The utility metric allows for the creation of MDPs with less knowledge of the domain. In chapter 9, we showed that experts can give the MDP method a boost by attempting the problems themselves, thereby “seeding” the MDP. We further loosened the problem knowledge constraints by introducing the BKB method (chapter 10), which allows hints to be generated from completely different problems. Finally, the method has been shown to be generalizable enough to be used with multiple tutors in different math and science domains (chapters 11). Specifically, we showed a tutor for Algebra and another for Stoichiometry. This means the door is open for others to automate ITS capabilities for CAI. In fact, we have already started to see this happening with other researchers citing and implementing our method, such as with the iList Linked List tutor (Fossati et al., 2009).

12.2 Impact

As more educational systems are being developed and more courses are moving online, a tremendous amount of data is being collected. The field of educational data mining (EDM) has begun to branch from the fields of intelligent tutoring and AI in education. With this new field of EDM our work is recognized as having a major impact. Our paper in the 9th International Conference on Intelligent Tutoring Systems in 2008 that covered the validation of the MDP method (Appendix A), was selected as a best paper nominee. Further, the MDP method has been included in the first Educational Data Mining Handbook, which is expected to be published in 2010 (Appendix A).

More importantly others have recognized our contribution and we have begun to see its impact in their work. Our work has been cited by three major ITS and EDM research groups (Chi 2009) (Fossati et al. 2009) (Fournier-Viger et al. 2009), and one of them implemented the MDP method into their tutor to successfully generate hints. The iList tutor is used to teach the computer science concept of linked lists (Fossati et al. 2009). The tutor represents linked lists in a graphical form that students can manipulate in order to gain a better understanding of the concepts without the need for large amounts of programming.

A screen shot of the tutor can be seen in Figure 12.2. Differing from our goal of providing hints, the group working on iList is using the method to deliver what they call "proactive feedback." This is essentially a forced hint message that appears when the student enters a certain state. The proactive feedback is positive or negative depending on the "goodness" value of the state the student has entered. If the MDP value is low the feedback may say that the student has a very low probability of solving the problem from where they are while a highly-valued state may deliver a message saying that the student is on the right track. This feedback was based on their experiments with human tutors and the feedback they provide to students working with linked lists.



Change the list L so that it represents [2, 3, 1, 8]

Welcome to iList - Version 0.60

Starting problem 1

Warning:
The list L has incorrect values.
You have lost one or more nodes.

```
Node *T;
T = new Node;
T->data = 3;
T->link = L->link;
L = T;
```

Warning:
The list L has incorrect values.
You have lost one or more nodes.

Figure 12.2. A screen shot of the iList tutor where the student is given a proactive hint based on a move made.

Fossati, et al., performed a ramp up experiment to see how well the method would be able to match states in various problems. Figure 11.6 shows the results of Problems 1 and 4. Problem 1 was the quickest learned problem and problem 4 was the slowest. Problem 1 hit 80% at just 4 attempts, 85% at 14 attempts, and 90% after 40 attempts. Problem 4 reached 80% after 52 attempts. These curves look very similar to the other problems we have seen, and show that the complexity of the problem and solution can affect the number of attempts needed to provide quality hints (or in this case proactive feedback). A classroom experiment was run to test learning gain. The five groups represent a human tutored group, three groups using the different versions of iList, and a control group that did an unrelated activity between the pre and post tests. Of the 3 versions of iList, only version 3 had the MDP method incorporated. The ANOVA results showed a significant difference across the five groups. The control group showed no significant learning and the human tutored group showed a 14% average gain. Although the difference between the 3 versions of iList was not significant, the MDP version was only marginally lower than the human tutored group with a 12% average learning gain compared to 8% and 10% in the other two iList groups. The authors believe that with a larger sample they may be able to show the version using the MDP method does lead to superior learning gain. They plan to run additional experiments with more students.

As adaptive computer aided instruction becomes more prevalent, techniques to automate the generation of these adaptive features will become more important. Due to the needs of the ITS community and the growth of data collection from computer aided instruction, we fully expect that in the next few years the methods described here will become common in the ITS field. The continued growth of the EDM community will also provide additional opportunities for these methods.

12.3 Future Work

We suggest extensions to our hint generation techniques to provide knowledge assessment, varied hint types, and user adaptation. This chapter concludes with a discussion of the future of using MDPs and other data-derived models for learning about and supporting student learning and problem solving.

There are a number of new directions we are exploring with the general MDP method (chapter 3). While we created only one MDP from a specific problem date set, it is possible to cluster different types of students and then create separate MDPs with different reward functions for each cluster. MDPs could be generated for 1) expert, 2) typical, and 3) least error-prone groups of students. The reward function we have described herein reflects an expert reward function, where the value for a state reflects the shortest path to the goal state. Alternatively, when the Hint Button is pressed, we could select a personalized reward function for the current student based on their student profile. If we have identified the student as an at-risk student, we may select the "least error-prone" reward function for generating hints. On the other hand, high-performing students would likely benefit from expert hints, while students between these two extremes may benefit from hints reflecting typical student behavior. If there is sufficient data, we can create separate MDPs for students in particular groups, such as high, low, and medium performers on a previous exercise, learning styles, GPA, or other factors, and use these to generate personalized hints. These hints would be contextualized both within the problem and by student characteristics.

The research on utility (chapter 8) introduced the idea of providing feedback based on a metric that measured the most frequent approach to a problem rather than the optimal solution that a teacher or proof solver might provide. It is often the case that students are not ready to apply optimal solutions to a problem. However, detecting this readiness is a challenging user modeling problem. It may be possible that student feedback based on frequency rather than optimality can provide most students with the help they need. Testing the standard MDP method, the utility method, seeded MDP method, and a solver all in a classroom setting could show interesting results. We believe that different type of hints may encourage differences in student reflection and engagement with the tutor.

The BKB method (chapter 10) can help increase the applicability of the hint factory set of tools. In the future we plan to incorporate the BKB into our existing MDP based tutor and perform experiments to see how often the two methods agree on hints, and how often the BKB can give hints that are not available via the traditional MDP method. We envision creating a framework for the Hint Factory that could be incorporated by other developers of existing CAI. This framework could be linked to a large data store of educational data such as the Pittsburgh Science of Learning Center's DataShop which could eventually allow for larger pooling of knowledge across multiple systems in the same domain.

This BKB work also lays the foundation for our research in automatic question generation. Generating questions for CAI is a very time consuming and difficult task. One of the most difficult aspects of question generation is the creation of different problems that are of the same difficulty and require the same skills. We believe that the use of our “values” with respect to the the novel model components (MCs) that we developed will provide a solution to this question generation problem. We plan to create an interface to allow educators to create several “equal” questions by using our BKB, then we will test with real students to see if these questions truly are equal based on expert and statistical analysis of students work.

Finally, we plan to team with other researchers in the EDM field, including the Pittsburgh Science of Learning Center DataShop (Koedinger et al 2008), to connect our methods automatically to large data sets that are currently being collected from a wide variety of domains. We plan not only to use our MDPs to provide hints, but also to give educators a visual tool to explore their problem spaces and understand the “value” of the individual steps in their problems. We believe that our work, along with the future directions we plan, will advance the cause of providing personalized learning to a much broader audience within a more diverse set of learning domains.

REFERENCES

Ainsworth, S. and Grimshaw, S. (2004) ‘Evaluating the REDEEM authoring tool: can teachers create effective learning environments?’, *International Journal of Artificial Intelligence in Education*, Vol. 14, pp.279–312.

Ainsworth, S.E., Major, N., Grimshaw, S.K., Hayes, M., Underwood, J.D., Williams, B. & Wood, D.J. (2003). REDEEM: Simple Intelligent Tutoring Systems From Usable Tools, In *T. Murray, S. Blessing & S.E. Ainsworth (eds). Advanced Tools for Advanced Technology Learning Environments.* pp. 205-232. Amsterdam: Kluwer Academic Publishers.

Ainsworth, S.E., Williams, B.C & Wood, D.J. (2001) Using the REDEEM ITS Authoring Environment in Naval Training. *IEEE International Conference on Advanced Learning Technologies*.

Ainsworth, S.E. Underwood, J.D. & Grimshaw, S.K. (2000) Using an ITS Authoring Tool to Explore Educators' Use of Instructional Strategies. In G. Gauthier, C. Frasson and K. VanLehn (Eds*.) Intelligent Tutoring Systems: Proceedings of the 5th International Conference ITS 2000*. pp 182-191.

Aleven, V., McLaren, B.M., Sewall, J., & Koedinger, K.R. (2009). A new paradigm for intelligent tutoring systems: Example-tracing tutors. *International Journal of Artificial Intelligence in Education*, 19(2), 105-154.

Aleven, V., McLaren, B. M., Sewall, J., & Koedinger, K. (2006). The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains. In M. Ikeda, K. D. Ashley, & T. W. Chan (Eds.), *Proceedings of the 8th International Conference on Intelligent Tutoring Systems* (ITS 2006), (pp. 61-70). Berlin: Springer Verlag.

Aleven, V., McLaren, B., Roll, I., & Koedinger, K. (2004). Toward tutoring help seeking: Applying cognitive modeling to meta-cognitive skills. In J. C. Lester, R. M. Vicari, & F. Paraguaçu (Eds.), *Proc. 7th Intl. Conference on Intelligent Tutoring Systems, ITS 2004* (pp. 227-239). Berlin: Springer Verlag.

Aleven, V., & Koedinger, K. R. (2000). Limitations of Student Control: Do Student Know when they need help? In *G. Gauthier, C. Frasson, & K. VanLehn (Eds.), Proceedings of the 5th International Conference on Intelligent Tutoring Systems,(ITS 2000)* (pp. 292-303). Berlin: Springer Verlag.

Anderson A., Skwarecki, (1986). The Automated Tutoring of Introductory Computer Programming. *Communications of the ACM* 29: pp. 842-849.

Anderson, J. R,, Boyle, C. F., & Reiser, B. J. (1985). Intelligent tutoring systems. Science, 228, 456-462.

Anderson, J. (1983). The Architecture of Cognition. Harvard University Press, Cambridge, MA.

Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences*, 4, 167-207.

Anderson, J. R., F. G. Conrad, and A. T. Corbett, (1989). Skill Acquisition and the LISP Tutor. *Cognitive Science*, 13, 467-505.

Antal P., Millinghoffer A. (2005). A probabilistic knowledge base using annotated Bayesian network features. *In Proceedings of the 6th International Symposium of Hungarian Researchers on Computational Intelligence*, pages 1–12.

Arroyo, I.,Woolf, B. (2005) Inferring learning and attitudes from a Bayesian Network of log file data. In C.K. Looie, G. McCalla, B. Bredeweg, and J. Breuker (eds) *Proceedings of the 12th International Conference on Artificial Intelligence in Education*. Amsterdam: IOS Press. pp 33-40.

Baker, R. (2007). Modeling and understanding students' off-task behavior in Intelligent Tutoring Systems. *CHI 2007 Proceedings.* San Jose, CA, USA.

Baker, R., Corbett, A.T., Aleven, V. (2008) More Accurate Student Modeling through Contextual Estimation of Slip and Guess Probabilities in Bayesian Knowledge Tracing. In E. Aimeur, & B. Woolf (Eds.) *Proceedings of the 9th International Conference on Intelligent Tutoring Systems (ITS 2008)*, pp. 406-415. Berlin, Germany: Springer Verlag.

Bass, E. J. (1998). "Towards an Intelligent Tutoring System for Situation AwarenessTraining in Complex, Dynamic Environments", *In B. Goettl, H.M. Halff, C.L. Redfield, and J.V. Shute (Eds) Proceed-ings of the 4th International Conference on Intelligent Tutoring Systems (ITS1998).* pp. 26-35. Springer., Berlin.

Beck, J., Woolf, B. P., and Beal, C. R. (2000). ADVISOR: A Machine Learning Architecture for Intelligent Tutor Construction. In: 7th National Conference on Artificial intelligence, pp. 552—557. AAAI Press / The MIT Press.

Bouma, W., Fudos, I., Hoffmann, C. M., Cai, J., AND Paige, R. (1995). A geometric constraint solver. Computer Aided Design. 27, 6 (June), 487–501.

Boyan, J., Freitag, D., Joachims, T. (1996). A machine learning architecture for optimizing web search engines. *In AAAI Workshop on Internet Based Information Systems*, August 1996. AAAI Press.

Buchanan, B.G. and Shortliffe, E.H. (eds) (1984). Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Reading, MA: Addison-Wesley,.

Burke, L.A. & Hutchins, H.M. (2007) Training transfer: An integrative literature review, *Human Resource Development Review*, 6, 3, 263-296.

Chang, K., Beck, J., Mostow, J., & Corbett, A. (2006). Does Help Help? A Bayes Net Approach to Modeling Tutor Interventions. *Proceedings of the AAAI2006 Workshop on Educational Data Mining*, 21st National Conference on Artificial Intelligence, Boston, MA.

Clancey, W. (1979). Transfer of Rule-Based Expertise through a Tutorial Dialogue. *Intelligent Tutoring Systems*, D. Sleeman and J. Brown. London, Academic Press: 201-225.

Conati, C., Gertner, A., VanLehn, K., and Druzdzel, M. (1997). Online student modeling for coached problem solving using Bayesian networks. *In Proceedings of the Sixth International Conference on User Modeling*, Sardinia, Italy, pages 231-242. User Modeling, Springer-Verlag.

Conati, C. Gertner, A. S. and VanLehn, K. (2002). Using Bayesian Networks to Manage Uncertainty in Student Modeling. In User Model. User-Adapt. Interact, volume 12 (4).

Craig S., D'Mello S., Witherspoon A., & Graesser A, (2007). Emote-Aloud during learning with AutoTutor: Applying the facial action coding system to cognitive-affective states during learning. *Cognition and Emotion*, 22, 777–788.

Croy, M. (1999). Graphic Interface Design and Deductive Proof Construction. Journal of Computers in Mathematics and Science Teaching, 18 (4):371–386.

D'Mello, S.K., Craig, S.D., Witherspoon, A., McDaniel, B., & Graesser, A.C. (2008). Automatic detection of learner's affect from conversational cues. *User Modeling and User-Adapted Interaction*, 18(1-2), 45-80.

Fossati, D., Di Eugenio, B., Ohlsson, S., Brown, c., Chen, L., Cosejo, D. (2009) I learn from you, you learn from me: How to make iList learn from students. In V. Dimitrova, R. Mizoguchi, B. Du Boulay and A. Graesser (Eds.), *Proceedings of the 14th International Conference on Artificial Intelligence in Education (AIED 2009)*. Springer-Verlag.

Fournier-Viger, P., Nkambou, R., Nguifo, E. (2009) Exploiting Partial Problem Spaces Learned from Users' Interactions to Provide Key Tutoring Services in Procedural and Ill-Defined Domains. In V. Dimitrova, R. Mizoguchi, B. Du Boulay and A. Graesser (Eds.), *Proceedings of the 14th International Conference on Artificial Intelligence in Education (AIED 2009)*. Springer-Verlag.

Graesser, A.C., Person, N.K. and Magliano, J.P., (1995). Collaborative dialogue patterns in naturalistic one-to-one tutoring. *Applied Cognitive Psychology* 9, pp. 495–522.

Haddawy, P. (1994) Generating Bayesian Networks from Probability Logic Knowledge Bases. *In Proceeding of the 10th Annual Conference on Uncertainty in Artificial Intelligence (UAI 1994)*. pp 262-269.

Heffernan, N.T. and Koedinger, K.R. (2002). An Intelligent Tutoring System Incorporating a Model of an Experienced Human Tutor. In Intelligent Tutoring Systems, pages 596–608.

Karabenick, S. A. (Ed.) (1998). Strategic help seeking. Implications for learning and teaching. Mahwah: Erlbaum.

Koedinger, K., Cunningham, K., Skogsholm A., and Leber, B. (2008). An open repository and analysis tools for fine-grained, longitudinal learner data. In R. Baker, T. Barnes, J. Beck (Eds.) *Proceedings of the 1st International Conference on Educational Data Mining (EDM 2008)*, pp. 157-166. Montreal, Canada.

Koedinger, K. R., Aleven, V., Heffernan. T., McLaren, B. & Hockenberry, M. (2004). Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. *In Proceedings of the 7th Intelligent Tutoring Systems Conference*, pp. 162-173.

Koedinger, K. R., Anderson, J.R., Hadley, W.H., & Mark, M . A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education,* 8, 30-43.

Koedinger, K. R ., & Sueker, E. L. F. (1996). PAT goes to college: Evaluating a cognitive tutor for developmental mathematics. In Proceedings of the Second International Conference on the Learning Sciences, (pp. 180-187). Charlottesville, VA: Association for the Advancement of Computing in Education.

Koedinger, K., and Anderson, J. (1995). Intelligent tutoring goes to the big city. Proc. of the International Conference on Artificial Intelligence in Education, Jim Greer (Ed). AACE: Charlottesville, VA pp. 421-428.

Landauer, T. K., Foltz, P. W., and Laham, D. (1998). Introduction to Latent Semantic Analysis. *Discourse Processes*, 25, 259-284.

Lynch, C., Ashley, K., Aleven, V., & Pinkwart, N. (2006). Defining Ill-Defined Domains; A literature survey. In V. Aleven, K. Ashley, C. Lynch, & N. Pinkwart (Eds.), *Proceedings of the Workshop on Intelligent Tutoring Systems for Ill-Defined Domains at the 8th International Conference on Intelligent Tutoring Systems* (p. 1-10).

Matsuda, N., & VanLehn, K. (2004). GRAMY: A geometry theorem prover capable of construction. *Journal of Automated Reasoning*, 32(1), 3-33.

Matsuda, N., Cohen, W. W., Sewall, J., Lacerda, G., & Koedinger, K. R. (2007). Predicting students performance with SimStudent that learns cognitive skills from observation. *In R. Luckin, K. R. Koedinger & J. Greer (Eds.), Proceedings of the international conference on Artificial Intelligence in Education* (pp. 467-476). Amsterdam, Netherlands: IOS Press.

McCune, W. and Shumsky, O. (2000) "Ivy: A Preprocessor and Proof Checker for First-order Logic". Chapter 16 in Computer-Aided Reasoning: ACL2 Case Studies (ed. M. Kaufmann, P. Manolios, and J Moore), Kluwer Academic Publishers.

McLaren, B., Koedinger, K., Schneider, M., Harrer, A., & Bollen, L. (2004). Bootstrapping Novice Data: Semi-automated tutor authoring using student log files, In Proc. Workshop on Analyzing Student-Tutor Interaction Logs to Improve Educational Outcomes, *In Proceedings of the 7th Intl. Conf. Intelligent Tutoring Systems (ITS-2004)* pp199-207.

McKendree, J. (1990). Effective Feedback Content for Tutoring Complex Skills. *Human-Computer Interaction* 5(4), pp. 381—413. Lawrence Earlbaum.

Merceron, A. and Yacef, K. (2005). Educational Data Mining: a Case Study. *In 12th International Conference on Artificial Intelligence in Education (AIED 2005)*, IOS Press.

Mitrovic, A., Martin, B., Suraweera, P. (2007). Constraint-based tutors: past, present and future. *IEEE Intelligent Systems, special issue on Intelligent Educational Systems*, vol. 22, no. 4, pp. 38-45.

Mitrovic, A., Suraweera, P., Martin, B., Zakharov, K., Milik, N., Holland, J. (2006). Authoring constraint-based tutors in ASPIRE. M. Ikeda, K. Ashley, and T.-W. Chan (Eds.) *Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS 2006)*, LNCS 4053, pp. 41-50.

Mitrovic, A., Koedinger, K. & Martin, B. (2003). A comparative analysis of cognitive tutoring and constraint-based modeling. *User Modeling*. pp 313-322.

Mitrovic, A., Martin, B., (2002). Evaluating the Effects of Open Student Models on Learning. *Second International Conference on Adaptive Hypermedia and Adaptive Web-Based System*s. Berlin, Springer-Verlag: 296-305.

Mitrovic, A., Ohlsson, S. (1999). Evaluation of a Constraint-Based Tutor for Database Language. *International Journal of Artificial Intelligence in Education* 11(2): pp 238-256.

Molnar, A. (1990). Computers in Education: A Historical Perspective of the Unfinished Task. *T.H.E. Journal* 18 (4).

Mostow, J., Beck, J. (2007). When the Rubber Meets the Road: Lessons from the In-School Adventures of an Automated Reading Tutor that Listens. In B. Schneider & S.-K. McDonald (Eds.), *Conceptualizing Scale-Up: Multidisciplinary Perspectives* (Vol. 2, pp. 183-200). Lanham, MD: Rowman & Littlefield.

Murphy, K., (2002). Dynamic Bayesian Networks: Representation, Inference and Learning. PhD thesis, UC Berkeley, Computer Science Division.

Murray, T. (1999). Authoring intelligent tutoring systems: An analysis of the state of the art. *Intl. J. Artificial Intelligence in Education*, pp10: 98-129.

Newell, Shaw and Simon (1957). "Empirical Explorations with the Logic Theory Machine", Allen Newell, J. C. Shaw, Herbert Simon, *Proceedings of the Western Joint Computer Conference 15*. pp. 218-329.

Nkambou, R., Mephu Nguifo, E., Fournier-Viger, P.(2008) Using Knowledge Discovery Techniques to Support Tutoring in an Ill-Defined Domain. In *E. Aimeur, & B. Woolf (Eds.) Intelligent Tutoring Systems (ITS 2008)*, pp. 395-405. Berlin: Springer Verlag.

Ohlsson, S. (1994). Constraint-Based Student Modeling. Student Modeling: The Key to Individualized Knowledge Based Instruction. JE Greer and G. McCalla: pp167-189.

Pavlik, P. I., Jr., & Anderson, J. R. (2004). An ACT-R model of memory applied to finding the optimal schedule of practice. *In Proceedings of the sixth International Conference on Cognitive Modeling* (pp. 376-377). Pittsburgh, PA: Carnegie Mellon University/University of Pittsburgh.

Rabiner. L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of IEEE*, pp. 257-286.

Ringenberg, M., VanLehn, K., (2006). "Scaffolding Problem Solving with Annotated Worked-Out Examples to Promote Deep Learning." In *Intelligent Tutoring Systems: Eighth International Conference (ITS 2006), Jhongli*, Taiwan. Springer-Verlag Lecture Notes in Computer Science.

Russell, S. and Norvig, P. (1995). Artificial Intelligence: A Modern Approach, Prentice Hall, Englewood Cliffs, NJ.

Salden, R., Aleven, V., Renkl, A., & Schwonke, R. (2008). Worked examples and tutored problem solving: redundant or synergistic forms of support? In C. Schunn (Ed.) *Proceedings of the 30th Annual Meeting of the Cognitive Science Society, CogSci 2008* (pp. 659-664). New York, NY: Lawrence Earlbaum.

Santos, E., Jr., & Santos, E. (1996) Bayesian Knowledge-bases. *Technical report AFIT/EN/TR96-05*, Department of Electrical and Computer Engineering, Air Force Institute of Technology.

Scheines, R., & Sieg, W. (1994). Computer environments for proof construction. *Interactive Learning Environments*, 4(2), 159-169.

Shute, V. J. (2008). Focus on formative feedback. *Review of Educational Research*, 78(1),153-189.

Sieg, W. (2007). The AProS project: Strategic thinking & computational logic. *Logic Journal of the IGPL*, 15(4):359–368.

Suppes, P. (1982). Future educational uses of interactive theorem proving. *University-Level Computer-Assisted Instruction at Stanford: 1968-1980.* Stanford Institute for Mathematical Studies in the Social Sciences: 339-430.

Sutton, S. and Barto, A.(1998). Reinforcement Learning: An Introduction. Cambridge: MIT Press,

Taatgen, N.A. & Anderson, J.R. (2008). *ACT-R*. In R. Sun (ed.), Constraints in Cognitive Architectures. Cambridge University Press, pp 170-185.

VanLehn, K. (2006). The behavior of tutoring systems, *International Journal of Artificial Intelligence in Education*, 16, pp: 227-265.

Vygotsky, L. (1986). *Thought and language*. Cambridge, MA: MIT Press.

APPENDIX A: DERIVED PUBLICATIONS

The following works were published by the author in conjunction with research in this thesis.

Barnes, T., Stamper, J. (2008). Toward Automatic Hint Generation for Logic Proof Tutoring Using Historical Student Data. *In E. Aimeur, & B. Woolf (Eds.) Proceedings of the 9th International Conference on Intelligent Tutoring Systems (ITS 2008)*, pp. 373-382. Berlin, Germany: Springer Verlag.

Barnes, T., Stamper, J., Croy, M., Lehman, L. (2008). A Pilot Study on Logic Proof Tutoring Using Hints Generated from Historical Student Data. In R. Baker, T. Barnes, J. Beck (Eds.) *Proceedings of the 1st International Conference on Educational Data Mining (EDM 2008)*, pp. 197-201. Montreal, Canada.

Barnes, T., Stamper, J. (2007). Toward the extraction of production rules for solving logic proofs. In C. Heiner, T. Barnes, & N. Heffernan (Eds.) *Proceedings of the 13th International Conference on Artificial Intelligence in Education, Educational Data Mining Workshop (AIED2007)*, pp. 11-20. Los Angeles, CA.

Barnes, T., Stamper, J., Madhyastha (2006). T. Comparative Analysis of Concept Derivation Using the Q-matrix Method and Facets. In J. Beck and T. Barnes (Eds.) *Proceedings of the AAAI 21th National Conference on Artificial Intelligence Educational Data Mining Workshop (AAAI2006)*, pp. 21-30. Menlo Park, CA: AAAI Press.

Croy, M., Barnes, T., and Stamper, J. (2008). Towards an Intelligent Tutoring System for Propositional Proof Construction. In A. Briggle, K. Waelbers, & E. Brey (Eds.) *Current Issues in Computing and Philosophy*, IOS Press: Amsterdam, Netherlands.

Stamper, J., Barnes, T. (2009). An Unsupervised, Frequency-based Metric for Selecting Hints in an MDP-based Tutor. In Barnes, T., Desmarais, M., Romero, C., & Ventura, S. (Eds.) *Educational Data Mining 2009: 2nd International Conference on Educational Data Mining, Proceedings.* pp. 180-189.Cordoba, Spain. July 1-3, 2009.

Stamper, J., Barnes, T. (2009). Utility in hint generation: Selection of hints from a corpus of student work. In V. Dimitrova, R. Mizoguchi, B. Du Boulay and A. Graesser (Eds.), *Proceedings of the 13th International Conference on Artificial Intelligence in Education (AIED 2009).* Brighton, England: IOS Press.

Stamper, J., Barnes, T. (2008). The Hint Factory: Automatic Generation of Contextualized Help for Existing Computer Aided Instruction. In G. Gouarderes, & R.M. Vicari (Eds.) *Proceedings of the 9th International Conference on Intelligent Tutoring Systems Young Researchers Track*, pp. 71-78. Montreal, Canada.

Stamper, J., Barnes, T., Croy, M., Lehmann, L. (2008). The Validity of Providing Automated Hints in an ITS Using a MDP. In Proceedings *of the 23nd AAAI Conference on Artificial Intelligence (AAAI 2008)*, pp. 1830-1831. Menlo Park, CA: AAAI Press.

Stamper, J., Barnes, T., Croy, M. (2007). Extracting Student Models for Intelligent Tutoring Systems. In Proceedings *of the 22nd AAAI Conference on Artificial Intelligence (AAAI 2007)*, pp. 1900-1901. Menlo Park, CA: AAAI Press.

Stamper, J. (2007). Automating the Generation of Student Models for Intelligent Tutoring Systems. In R. Luckin, K. Koedinger, & J. Greer (Eds.), *Proceedings of the 13th International Conference on Artificial Intelligence in Education (AIED 2007).* pp. 701-702. Amsterdam, the Netherlands: IOS Press.

Stamper, J. (2006). Automating the Generation of Production Rules for Intelligent Tutoring Systems. *In Proceedings of the 9th International Conference on Interactive Computer Aided Learning (ICL2006)*, Kassel University Press.

APPENDIX B: PROBLEM DESCRIPTIONS AND SYMBOL NOTATION

Many different forms of notation exist in logic. In this research we use a symbol notation. The notation and English equivalents can be seen in Figure A.1.

Table A.1. Symbol notation and English Eqivalents

Symbol	English	Symbolic Example	English Example
~	Not	~A	Not A
>	If/Then	A>B	If A then B
v	Or	AvB	A or B
&	And	A&B	A and B
=	Implies	A=B	A Implies B

As described in the main chapters, the logic proof problems in this research are from two different computer aided instruction used to teach logic. The NCSU proof tutor is a text based system where students are given a list of premises and a conclusion. The students input new statements and actions along with justifications of each line. A list of the NCSU problem set can be seen in A.2 and a list of the actions available for use in the NCSU problem set can be seen in A.3.

Table A.2. Problem Descriptions for NCSU Dataset

Problem	Givens	Conclusion
1	A>B, C>D, ~(A>D)	B&~C
2	A>B, ~C>D, ~Bv~D	A>C
3	(BvA)>C	A>(B>C)
4	Av(B>C), BvC, C>A	A

Table A.3. Actions in NCSU Proof Tutor

Action	Description	Number Used in Tutor
1	$[(p \rightarrow q) \wedge (q \rightarrow r)] \Rightarrow (p \rightarrow r)$	Hypothetical Syllogism
2	$[(p \vee q) \wedge \neg p] \Rightarrow q$	Disjunctive Syllogism
3	$[p \wedge (p \rightarrow q)] \Rightarrow q$	Modus Ponens
4	$[(p \rightarrow q) \wedge \neg q] \Rightarrow \neg p$	Modus Tollens
5	$(p \rightarrow 0) \Rightarrow \neg p$	Absurdity
6	$(p \wedge q) \Rightarrow p$	Simplification
7	$[(p \rightarrow q) \wedge (r \rightarrow s)] \Rightarrow [(p \vee r) \rightarrow (q \vee s)]$	Constructive Dilemma ($\vee$)
8	$[(p \rightarrow q) \wedge (r \rightarrow s)] \Rightarrow [(p \wedge r) \rightarrow (q \wedge s)]$	Constructive Dilemma ($\wedge$)
9	$p \Rightarrow (p \vee q)$	Addition
10	$[p \wedge q] \Rightarrow (p \wedge q)$	Conjunction
11	$\neg \neg p \Leftrightarrow p$	Double Negation
12	$(p \vee q) \Leftrightarrow (q \vee p)$	Commutative Laws
13	$(p \wedge q) \Leftrightarrow (q \wedge p)$	Commutative Laws
14	$[(p \vee q) \vee r] \Leftrightarrow [p \vee (q \vee r)]$	Associative Laws
15	$[(p \wedge q) \wedge r] \Leftrightarrow [p \wedge (q \wedge r)]$	Associative Laws
16	$[p \vee (q \wedge r)] \Leftrightarrow [(p \vee q) \wedge (p \vee r)]$	Distributive Laws
17	$[p \wedge (q \vee r)] \Leftrightarrow [(p \wedge q) \vee (p \wedge r)]$	Distributive Laws
18	$(p \vee p) \Leftrightarrow p$	Idempotent Laws
19	$(p \wedge p) \Leftrightarrow p$	Idempotent Laws
20	$(p \vee 0) \Leftrightarrow p$	Idempotent Laws
21	$(p \vee 1) \Leftrightarrow 1$	Idempotent Laws
22	$(p \wedge 0) \Leftrightarrow 0$	Identity Laws
23	$(p \wedge 1) \Leftrightarrow p$	Identity Laws
24	$(p \rightarrow p) \Leftrightarrow 1$	Identity Laws
25	$(p \vee \neg p) \Leftrightarrow 1$	Tautology
26	$(p \wedge \neg p) \Leftrightarrow 0$	Contradiction
27	$\neg(p \vee q) \Leftrightarrow (\neg p \wedge \neg q)$	DeMorgans
28	$\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)$	DeMorgans
29	$(p \rightarrow q) \Leftrightarrow (\neg q \rightarrow \neg p)$	Contrapositive
30	$(p \rightarrow q) \Leftrightarrow (\neg p \vee q)$	Contrapositive
31	$(p \rightarrow q) \Leftrightarrow \neg (p \wedge \neg q)$	Implication
32	$[(p \rightarrow r) \wedge (q \rightarrow r)] \Leftrightarrow [(p \vee q) \rightarrow r]$	Implication
33	$[(p \rightarrow q) \wedge (p \rightarrow r)] \Leftrightarrow [p \rightarrow (q \wedge r)]$	Implication

The Deep Thought tutor has a graphical interface which was described in detail in chapter 3. This tutor can be set to display the problems using the symbol or English notation. The default is set based on the instructor preference, although students can change the notation type at the beginning of a problem. The problems used in this research from Deep Thought can be seen in A.4, and a list of actions available in Deep Thought can be seen in A.5.

Table A.4. Problem Descriptions for DT Dataset

Problem	Givens	Conclusion
1-1	A>(B&C), AvD, ~D&E	B
1-2	(FvG)>H, IvF, ~I&J	H
1-4	(~T&S)>~R, ~T, (~QvP)>S, Q>T	~RvN
1-5	Z>(~Y>X), Z&~W, Wv(T>S), ~YvT	XvS
3-2	(A>~B)vC, ~C,DvB	~D>~A
3-5	K>M, Z>R, ~(K>R)	M&~Z
3-6	~(T&L), ~T>~N, ~(EvT)	~N
3-8	Y=P, ~Y>~C, ~P=~C	Y>C

Table A.5. Actions available for DT Dataset

Action	Description
MP	Modus Ponens
MT	Modus Tollens
DS	Disjunctive Syllogism
HS	Hypothetical Syllogism
ADD	Addition
SIMP	Simplification
CONJ	Conjunction
CD	Constructive Dilema
DN	Double Negation
DEM	Demorgan’s
IMPL	Implication
TRANS	Transitive
COM	Communication
ASSOC	Association
DIST	Distribution
ABS	Absorption
EXP	Exportation
TAUT	Tautology

APPENDIX C: GENERATED MARKOV GRAPHS

The following is an example Markov Descision Processes created for the research from the NCSU data set problem number 1. Additional MDPs are available from the author who can be contacted at john@stamper.org.

```
State: S1   a>b,c>d,-(a>d)
            Action: 1    To State: 769   Prob: 0.169204737732657   Times: 1
            Action: 1    To State: 653   Prob: 0.169204737732657   Times: 1
            Action: 1    To State: 603   Prob: 0.169204737732657   Times: 1
            Action: 1    To State: 537   Prob: 0.169204737732657   Times: 1
            Action: 1    To State: 467   Prob: 0.169204737732657   Times: 1
            Action: 1    To State: 325   Prob: 0.169204737732657   Times: 1
            Action: 1    To State: 122   Prob: 0.169204737732657   Times: 1
            Action: 1    To State: 114   Prob: 0.169204737732657   Times: 1
            Action: 1    To State: 113   Prob: 1.52284263959391    Times: 9
            Action: 2    To State: 571   Prob: 0.169204737732657   Times: 1
            Action: 2    To State: 311   Prob: 0.169204737732657   Times: 1
            Action: 2    To State: 33    Prob: 0.169204737732657   Times: 1
            Action: 3    To State: 811   Prob: 0.169204737732657   Times: 1
            Action: 3    To State: 785   Prob: 0.169204737732657   Times: 1
            Action: 3    To State: 783   Prob: 0.169204737732657   Times: 1
            Action: 3    To State: 573   Prob: 0.169204737732657   Times: 1
            Action: 3    To State: 528   Prob: 0.676818950930626   Times: 4
            Action: 3    To State: 466   Prob: 0.169204737732657   Times: 1
            Action: 3    To State: 462   Prob: 0.169204737732657   Times: 1
            Action: 3    To State: 303   Prob: 0.169204737732657   Times: 1
            Action: 3    To State: 236   Prob: 0.50761421319797    Times: 3
            Action: 4    To State: 570   Prob: 0.169204737732657   Times: 1
            Action: 4    To State: 569   Prob: 0.169204737732657   Times: 1
            Action: 4    To State: 529   Prob: 0.338409475465313   Times: 2
            Action: 4    To State: 33    Prob: 0.169204737732657   Times: 1
            Action: 5    To State: 770   Prob: 0.169204737732657   Times: 1
            Action: 5    To State: 149   Prob: 0.169204737732657   Times: 1
            Action: 7    To State: 588   Prob: 0.169204737732657   Times: 1
            Action: 7    To State: 402   Prob: 0.846023688663283   Times: 5
            Action: 7    To State: 330   Prob: 0.169204737732657   Times: 1
            Action: 7    To State: 150   Prob: 0.676818950930626   Times: 4
            Action: 8    To State: 331   Prob: 0.50761421319797    Times: 3
                                                                   Times:
            Action: 8    To State: 72    Prob: 1.69204737732657    10
            Action: 9    To State: 615   Prob: 0.169204737732657   Times: 1
            Action: 10   To State: 527   Prob: 0.169204737732657   Times: 1
            Action: 27   To State: 180   Prob: 1.01522842639594    Times: 6
            Action: 27   To State: 108   Prob: 0.169204737732657   Times: 1
            Action: 28   To State: 290   Prob: 0.676818950930626   Times: 4
```

Action: 29	To State: 761	Prob: 0.169204737732657	Times: 1
Action: 29	To State: 672	Prob: 0.169204737732657	Times: 1
Action: 29	To State: 362	Prob: 0.169204737732657	Times: 1
Action: 29	To State: 252	Prob: 0.676818950930626	Times: 4
Action: 29	To State: 246	Prob: 1.35363790186125	Times: 8
Action: 29	To State: 53	Prob: 2.7072758037225	Times: 16
Action: 29	To State: 19	Prob: 1.86125211505922	Times: 11
Action: 30	To State: 826	Prob: 0.169204737732657	Times: 1
Action: 30	To State: 807	Prob: 0.169204737732657	Times: 1
Action: 30	To State: 705	Prob: 0.169204737732657	Times: 1
Action: 30	To State: 694	Prob: 0.338409475465313	Times: 2
Action: 30	To State: 658	Prob: 0.338409475465313	Times: 2
Action: 30	To State: 583	Prob: 0.338409475465313	Times: 2
Action: 30	To State: 564	Prob: 0.169204737732657	Times: 1
Action: 30	To State: 543	Prob: 0.338409475465313	Times: 2
Action: 30	To State: 540	Prob: 0.338409475465313	Times: 2
Action: 30	To State: 536	Prob: 0.676818950930626	Times: 4
Action: 30	To State: 494	Prob: 0.676818950930626	Times: 4
Action: 30	To State: 239	Prob: 1.01522842639594	Times: 6
Action: 30	To State: 236	Prob: 1.86125211505922	Times: 11
Action: 30	To State: 191	Prob: 0.338409475465313	Times: 2
Action: 30	To State: 148	Prob: 0.169204737732657	Times: 1
Action: 30	To State: 128	Prob: 0.50761421319797	Times: 3
Action: 30	To State: 81	Prob: 0.169204737732657	Times: 1
Action: 30	To State: 34	Prob: 1.69204737732657	Times: 10
Action: 30	To State: 33	Prob: 0.676818950930626	Times: 4
Action: 30	To State: 7	Prob: 14.2131979695431	Times: 84
Action: 30	To State: 2	Prob: 38.917089678511	Times: 230
Action: 31	To State: 784	Prob: 0.169204737732657	Times: 1
Action: 31	To State: 716	Prob: 0.169204737732657	Times: 1
Action: 31	To State: 468	Prob: 0.169204737732657	Times: 1
Action: 31	To State: 345	Prob: 0.338409475465313	Times: 2
Action: 31	To State: 302	Prob: 0.169204737732657	Times: 1
Action: 31	To State: 281	Prob: 0.169204737732657	Times: 1
Action: 31	To State: 280	Prob: 0.169204737732657	Times: 1
Action: 31	To State: 210	Prob: 1.69204737732657	Times: 10
Action: 31	To State: 143	Prob: 0.846023688663283	Times: 5
Action: 31	To State: 115	Prob: 4.23011844331641	Times: 25
Action: 31	To State: 85	Prob: 0.338409475465313	Times: 2
Action: 31	To State: 40	Prob: 0.169204737732657	Times: 1
Action: 31	To State: 16	Prob: 7.27580372250423	Times: 43
Action: 32	To State: 753	Prob: 0.169204737732657	Times: 1
Action: 32	To State: 210	Prob: 0.169204737732657	Times: 1

```
              Action: 37        To State: 535     Prob: 0.169204737732657        Times: 1

State: S2     a>b,c>d,-(a>d),-(-a+d)
              Action: 1         To State: 126     Prob: 0.3690036900369          Times: 1
              Action: 2         To State: 126     Prob: 0.3690036900369          Times: 1
              Action: 3         To State: 501     Prob: 0.3690036900369          Times: 1
              Action: 3         To State: 266     Prob: 0.738007380073801        Times: 2
              Action: 27        To State: 636     Prob: 0.3690036900369          Times: 1
              Action: 27        To State: 541     Prob: 0.3690036900369          Times: 1
              Action: 27        To State: 484     Prob: 0.3690036900369          Times: 1
              Action: 27        To State: 475     Prob: 1.1070110701107          Times: 3
              Action: 27        To State: 127     Prob: 1.4760147601476          Times: 4
                                                                                 Times:
              Action: 27        To State: 71      Prob: 5.16605166051661         14
              Action: 27        To State: 51      Prob: 0.738007380073801        Times: 2
                                                                                 Times:
              Action: 27        To State: 3       Prob: 71.5867158671587         194
                                                                                 Times:
              Action: 28        To State: 3       Prob: 14.0221402214022         38
              Action: 29        To State: 710     Prob: 0.3690036900369          Times: 1
              Action: 29        To State: 3       Prob: 0.738007380073801        Times: 2
              Action: 30        To State: 616     Prob: 0.3690036900369          Times: 1
              Action: 30        To State: 64      Prob: 1.1070110701107          Times: 3
              Action: 31        To State: 794     Prob: 0.3690036900369          Times: 1

State: S3     a>b,c>d,-(a>d),-(-a+d),a*-d
              Action: 1         To State: 504     Prob: 0.304878048780488        Times: 1
              Action: 1         To State: 502     Prob: 0.609756097560976        Times: 2
              Action: 1         To State: 26      Prob: 0.609756097560976        Times: 2
              Action: 2         To State: 27      Prob: 0.304878048780488        Times: 1
              Action: 3         To State: 611     Prob: 0.609756097560976        Times: 2
              Action: 3         To State: 399     Prob: 1.52439024390244         Times: 5
              Action: 3         To State: 183     Prob: 0.609756097560976        Times: 2
                                                                                 Times:
              Action: 3         To State: 27      Prob: 15.2439024390244         50
                                                                                 Times:
              Action: 3         To State: 4       Prob: 43.5975609756098         143
              Action: 4         To State: 610     Prob: 1.21951219512195         Times: 4
              Action: 4         To State: 182     Prob: 0.609756097560976        Times: 2
                                                                                 Times:
              Action: 4         To State: 171     Prob: 9.14634146341463         30
                                                                                 Times:
              Action: 4         To State: 170     Prob: 5.18292682926829         17
              Action: 4         To State: 27      Prob: 0.609756097560976        Times: 2
              Action: 5         To State: 399     Prob: 0.304878048780488        Times: 1
              Action: 7         To State: 26      Prob: 0.609756097560976        Times: 2
              Action: 8         To State: 164     Prob: 1.21951219512195         Times: 4
              Action: 10        To State: 508     Prob: 0.609756097560976        Times: 2
              Action: 27        To State: 182     Prob: 0.304878048780488        Times: 1
              Action: 28        To State: 26      Prob: 0.304878048780488        Times: 1
                                                                                 Times:
              Action: 29        To State: 91      Prob: 3.35365853658537         11
```

```
             Action: 30     To State: 685    Prob: 0.609756097560976    Times: 2
             Action: 30     To State: 605    Prob: 0.304878048780488    Times: 1
             Action: 30     To State: 6      Prob: 0.304878048780488    Times: 1
             Action: 30     To State: 269    Prob: 0.304878048780488    Times: 1
                                                                        Times:
             Action: 30     To State: 173    Prob: 4.57317073170732     15
                                                                        Times:
             Action: 30     To State: 140    Prob: 3.65853658536585     12
             Action: 31     To State: 782    Prob: 0.304878048780488    Times: 1
             Action: 31     To State: 709    Prob: 0.304878048780488    Times: 1
             Action: 31     To State: 469    Prob: 0.304878048780488    Times: 1
             Action: 31     To State: 332    Prob: 0.609756097560976    Times: 2
             Action: 31     To State: 268    Prob: 0.914634146341463    Times: 3
             Action: 31     To State: 49     Prob: 0.304878048780488    Times: 1
             Action: 32     To State: 27     Prob: 0.304878048780488    Times: 1
             Action: 33     To State: 163    Prob: 0.304878048780488    Times: 1

State: S4    a>b,c>d,-(a>d),-(-a+d),a*-d,b
             Action: 2      To State: 52     Prob: 0.50251256281407     Times: 1
             Action: 3      To State: 52     Prob: 2.01005025125628     Times: 4
                                                                        Times:
             Action: 4      To State: 52     Prob: 16.5829145728643     33
                                                                        Times:
             Action: 4      To State: 5      Prob: 52.2613065326633     104
             Action: 5      To State: 28     Prob: 0.50251256281407     Times: 1
             Action: 10     To State: 400    Prob: 1.50753768844221     Times: 3
             Action: 29     To State: 405    Prob: 1.00502512562814     Times: 2
                                                                        Times:
             Action: 29     To State: 119    Prob: 11.0552763819095     22
             Action: 29     To State: 118    Prob: 3.51758793969849     Times: 7
                                                                        Times:
             Action: 29     To State: 52     Prob: 6.03015075376884     12
                                                                        Times:
             Action: 30     To State: 198    Prob: 5.0251256281407      10

State: S5    a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c
             Action: 2      To State: 408    Prob: 0.854700854700855    Times: 1
             Action: 10     To State: 631    Prob: 0.854700854700855    Times: 1
             Action: 10     To State: 403    Prob: 0.854700854700855    Times: 1
             Action: 10     To State: 274    Prob: 7.69230769230769     Times: 9
                                                                        Times:
             Action: 10     To State: 6      Prob: 89.7435897435898     105

State: S6    b*-c
             Action: 10     To State: 361    Prob: 9.09090909090909     Times: 1
             Action: 10     To State: 356    Prob: 18.1818181818182     Times: 2
             Action: 10     To State: 6      Prob: 72.7272727272727     Times: 8

             a>b,c>d,-(a>d),-
State: S7    a+b
             Action: 2      To State: 601    Prob: 2.08333333333333     Times: 2
             Action: 27     To State: 376    Prob: 1.04166666666667     Times: 1
```

```
                Action: 28        To State: 377     Prob: 1.04166666666667     Times: 1
                Action: 29        To State: 627     Prob: 1.04166666666667     Times: 1
                Action: 30        To State: 745     Prob: 2.08333333333333     Times: 2
                Action: 30        To State: 563     Prob: 1.04166666666667     Times: 1
                Action: 30        To State: 559     Prob: 1.04166666666667     Times: 1
                Action: 30        To State: 558     Prob: 1.04166666666667     Times: 1
                Action: 30        To State: 275     Prob: 1.04166666666667     Times: 1
                Action: 30        To State: 137     Prob: 3.125                Times: 3
                Action: 30        To State: 123     Prob: 2.08333333333333     Times: 2
                Action: 30        To State: 43      Prob: 13.5416666666667     Times: 13
                Action: 30        To State: 42      Prob: 1.04166666666667     Times: 1
                Action: 30        To State: 41      Prob: 1.04166666666667     Times: 1
                Action: 30        To State: 8       Prob: 65.625               Times: 63
                Action: 31        To State: 291     Prob: 2.08333333333333     Times: 2

State: S8       a>b,c>d,-(a>d),-a+b,-c+d
                Action: 3         To State: 435     Prob: 1.33333333333333     Times: 1
                Action: 4         To State: 827     Prob: 1.33333333333333     Times: 1
                Action: 10        To State: 655     Prob: 1.33333333333333     Times: 1
                Action: 10        To State: 654     Prob: 1.33333333333333     Times: 1
                Action: 27        To State: 168     Prob: 1.33333333333333     Times: 1
                Action: 28        To State: 353     Prob: 1.33333333333333     Times: 1
                Action: 29        To State: 436     Prob: 2.66666666666667     Times: 2
                Action: 29        To State: 168     Prob: 1.33333333333333     Times: 1
                Action: 30        To State: 516     Prob: 1.33333333333333     Times: 1
                Action: 30        To State: 31      Prob: 1.33333333333333     Times: 1
                Action: 30        To State: 30      Prob: 1.33333333333333     Times: 1
                Action: 30        To State: 29      Prob: 1.33333333333333     Times: 1
                Action: 30        To State: 10      Prob: 70.6666666666667     Times: 53
                Action: 30        To State: 9       Prob: 4                    Times: 3
                Action: 31        To State: 641     Prob: 2.66666666666667     Times: 2
                Action: 31        To State: 213     Prob: 5.33333333333333     Times: 4

State: S9       a>b,c>d,-(a>d),-a+b,-c+d,-(-a+b) 0

State: S10      a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d)
                Action: 1         To State: 728     Prob: 1.58730158730159     Times: 1
                Action: 27        To State: 354     Prob: 3.17460317460317     Times: 2
                Action: 27        To State: 129     Prob: 7.93650793650794     Times: 5
                Action: 27        To State: 12      Prob: 76.1904761904762     Times: 48
                Action: 27        To State: 11      Prob: 6.34920634920635     Times: 4
                Action: 28        To State: 12      Prob: 4.76190476190476     Times: 3

State: S11      a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d 0

State: S12      a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d
                Action: 1         To State: 228     Prob: 1.26582278481013     Times: 1
                Action: 1           To State: 227     Prob: 1.26582278481013      Times: 1
                Action: 1           To State: 226     Prob: 1.26582278481013      Times: 1
                Action: 1           To State: 130     Prob: 2.53164556962025      Times: 2
                Action: 2           To State: 138     Prob: 8.86075949367089      Times: 7
                Action: 2           To State: 124     Prob: 8.86075949367089      Times: 7
                                                                                  Times:
                Action: 2           To State: 14      Prob: 20.253164556962       16
                                                                                  Times:
                Action: 2           To State: 13      Prob: 20.253164556962       16
                                                                                  Times:
                Action: 3           To State: 14      Prob: 12.6582278481013      10
                Action: 4           To State: 167     Prob: 1.26582278481013      Times: 1
                Action: 5           To State: 758     Prob: 1.26582278481013      Times: 1
                Action: 7           To State: 131     Prob: 1.26582278481013      Times: 1
                Action: 8           To State: 258     Prob: 1.26582278481013      Times: 1
                Action: 8           To State: 257     Prob: 1.26582278481013      Times: 1
                Action: 8           To State: 13      Prob: 1.26582278481013      Times: 1
                Action: 9           To State: 125     Prob: 1.26582278481013      Times: 1
                Action: 10          To State: 6       Prob: 1.26582278481013      Times: 1
                Action: 10          To State: 470     Prob: 2.53164556962025      Times: 2
                Action: 27          To State: 758     Prob: 2.53164556962025      Times: 2
                Action: 27          To State: 169     Prob: 2.53164556962025      Times: 2
                Action: 27          To State: 155     Prob: 1.26582278481013      Times: 1
                Action: 27          To State: 154     Prob: 1.26582278481013      Times: 1
                Action: 29          To State: 471     Prob: 1.26582278481013      Times: 1
                Action: 31          To State: 156     Prob: 1.26582278481013      Times: 1

State: S13      a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b 0

State: S14      a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b
                                                                                  Times:
                Action: 2           To State: 15      Prob: 51.6129032258064      16
                Action: 3           To State: 15      Prob: 3.2258064516129       Times: 1
                Action: 4           To State: 719     Prob: 3.2258064516129       Times: 1
                Action: 4           To State: 718     Prob: 3.2258064516129       Times: 1
                Action: 4           To State: 240     Prob: 9.67741935483871      Times: 3
                Action: 4           To State: 15      Prob: 22.5806451612903      Times: 7
                Action: 7           To State: 59      Prob: 3.2258064516129       Times: 1
                Action: 9           To State: 721     Prob: 3.2258064516129       Times: 1

State: S15      a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,-c
                Action: 2           To State: 550     Prob: 4.16666666666667      Times: 1
                Action: 9           To State: 32      Prob: 4.16666666666667      Times: 1
                                                                                  Times:
                Action: 10          To State: 6       Prob: 91.6666666666667      22

State: S16      a>b,c>d,-(a>d),(a*-d)
                Action: 1           To State: 622     Prob: 1.58730158730159      Times: 1
                Action: 1           To State: 364     Prob: 1.58730158730159      Times: 1
                Action: 1           To State: 109     Prob: 1.58730158730159      Times: 1
                Action: 3           To State: 706     Prob: 1.58730158730159      Times: 1
```

```
            Action: 3       To State: 110   Prob: 7.93650793650794    Times: 5
                                                                      Times:
            Action: 3       To State: 17    Prob: 31.7460317460317    20
            Action: 4       To State: 517   Prob: 1.58730158730159    Times: 1
            Action: 4       To State: 82    Prob: 11.1111111111111    Times: 7
            Action: 5       To State: 720   Prob: 1.58730158730159    Times: 1
            Action: 5       To State: 364   Prob: 1.58730158730159    Times: 1
            Action: 8       To State: 392   Prob: 1.58730158730159    Times: 1
            Action: 8       To State: 286   Prob: 3.17460317460317    Times: 2
            Action: 8       To State: 285   Prob: 3.17460317460317    Times: 2
            Action: 8       To State: 93    Prob: 3.17460317460317    Times: 2
            Action: 9       To State: 312   Prob: 1.58730158730159    Times: 1
            Action: 27      To State: 420   Prob: 1.58730158730159    Times: 1
            Action: 29      To State: 368   Prob: 6.34920634920635    Times: 4
            Action: 30      To State: 802   Prob: 1.58730158730159    Times: 1
            Action: 30      To State: 690   Prob: 1.58730158730159    Times: 1
            Action: 30      To State: 312   Prob: 1.58730158730159    Times: 1
            Action: 31      To State: 481   Prob: 3.17460317460317    Times: 2
            Action: 31      To State: 111   Prob: 1.58730158730159    Times: 1
            Action: 31      To State: 94    Prob: 6.34920634920635    Times: 4
            Action: 35      To State: 109   Prob: 1.58730158730159    Times: 1

State: S17  a>b,c>d,-(a>d),(a*-d),b
            Action: 3       To State: 50    Prob: 3.44827586206897    Times: 1
            Action: 4       To State: 50    Prob: 13.7931034482759    Times: 4
                                                                      Times:
            Action: 4       To State: 18    Prob: 51.7241379310345    15
            Action: 5       To State: 365   Prob: 3.44827586206897    Times: 1
            Action: 9       To State: 578   Prob: 3.44827586206897    Times: 1
            Action: 10      To State: 578   Prob: 6.89655172413793    Times: 2
            Action: 10      To State: 6     Prob: 3.44827586206897    Times: 1
            Action: 29      To State: 397   Prob: 3.44827586206897    Times: 1
            Action: 29      To State: 396   Prob: 3.44827586206897    Times: 1
            Action: 30      To State: 779   Prob: 3.44827586206897    Times: 1
            Action: 31      To State: 579   Prob: 3.44827586206897    Times: 1

State: S18  a>b,c>d,-(a>d),(a*-d),b,-c
            Action: 9       To State: 366   Prob: 5.88235294117647    Times: 1
                                                                      Times:
            Action: 10      To State: 6     Prob: 94.1176470588235    16

State: S19  a>b,c>d,-(a>d),-(-d>-a)
            Action: 29      To State: 434   Prob: 10                  Times: 1
            Action: 29      To State: 298   Prob: 10                  Times: 1
            Action: 29      To State: 297   Prob: 10                  Times: 1
            Action: 30      To State: 20    Prob: 70                  Times: 7

State: S20  a>b,c>d,-(a>d),-(-d>-a),-(d+-a)
            Action: 27      To State: 581   Prob: 12.5                Times: 1
            Action: 27      To State: 458   Prob: 25                  Times: 2
            Action: 27      To State: 21    Prob: 62.5                Times: 5
```

```
State: S21    a>b,c>d,-(a>d),-(-d>-a),-(d+-a),a*-d
              Action: 29        To State: 22     Prob: 100                        Times: 5

State: S22    a>b,c>d,-(a>d),-(-d>-a),-(d+-a),a*-d,-d>-c
              Action: 3         To State: 90     Prob: 28.5714285714286           Times: 2
              Action: 3         To State: 23     Prob: 71.4285714285714           Times: 5

State: S23    a>b,c>d,-(a>d),-(-d>-a),-(d+-a),a*-d,-d>-c,-c
              Action: 3         To State: 25     Prob: 62.5                       Times: 5
              Action: 3         To State: 24     Prob: 25                         Times: 2
              Action: 10        To State: 24     Prob: 12.5                       Times: 1

State: S24    a>b,c>d,-(a>d),-(-d>-a),-(d+-a),a*-d,-d>-c,-c,b 0

State: S25    a>b,c>d,-(a>d),-(-d>-a),-(d+-a),a*-d,-d>-c,-c,b
              Action: 10        To State: 6      Prob: 100                        Times: 5

State: S26    a>b,c>d,-(a>d),-(-a+d),a*-d,a 0

State: S27    a>b,c>d,-(a>d),-(-a+d),a*-d,b 0

State: S28    a>b,c>d,-(a>d),-(-a+d),a*-d,b,-d 0

State: S29    a>b,c>d,-(a>d),-a+b,-c+d,a*-b 0

State: S30    a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d) 0

State: S31    a>b,c>d,-(a>d),-a+b,-c+d,--a+d 0

State: S32    a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,-c,b*-c 0

State: S33    a>b,c>d,-(a>d),-(-a+d) 0

              a>b,c>d,-(a>d),-
State: S34    c+d
              Action: 9         To State: 748    Prob: 9.09090909090909           Times: 1
              Action: 28        To State: 669    Prob: 9.09090909090909           Times: 1
              Action: 30        To State: 184    Prob: 18.1818181818182           Times: 2
              Action: 30        To State: 35     Prob: 63.6363636363636           Times: 7

State: S35    a>b,c>d,-(a>d),-c+d,-(-a+d)
              Action: 27        To State: 36     Prob: 85.7142857142857           Times: 6
              Action: 28        To State: 36     Prob: 14.2857142857143           Times: 1

State: S36    a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d
              Action: 1         To State: 750    Prob: 8.33333333333333           Times: 1
              Action: 2         To State: 193    Prob: 8.33333333333333           Times: 1
              Action: 3         To State: 363    Prob: 16.6666666666667           Times: 2
              Action: 3         To State: 216    Prob: 33.3333333333333           Times: 4
              Action: 4         To State: 749    Prob: 8.33333333333333           Times: 1
```

```
             Action: 7          To State: 192     Prob: 8.33333333333333          Times: 1
             Action: 28         To State: 38      Prob: 8.33333333333333          Times: 1
             Action: 28         To State: 37      Prob: 8.33333333333333          Times: 1

State: S37   a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d,-(d+-a) 0

State: S38   a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d,-(d+-a)
             Action: 30         To State: 39      Prob: 100                       Times: 1

State: S39   a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d,-(d+-a),-(-d>-a)

State: S40   a>b,c>d,-(a>d),a*-b 0

State: S41   a>b,c>d,-(a>d),-a+b,-(a+-d) 0

State: S42   a>b,c>d,-(a>d),-a+b,-a+d 0

State: S43   a>b,c>d,-(a>d),-a+b,-(-a+d)
             Action: 27         To State: 44      Prob: 85.7142857142857          Times: 6
             Action: 28         To State: 357     Prob: 14.2857142857143          Times: 1

State: S44   a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d
             Action: 1          To State: 6       Prob: 6.25                      Times: 1
             Action: 1          To State: 747     Prob: 6.25                      Times: 1
             Action: 1          To State: 746     Prob: 6.25                      Times: 1
             Action: 1          To State: 409     Prob: 6.25                      Times: 1
             Action: 2          To State: 358     Prob: 12.5                      Times: 2
             Action: 3          To State: 358     Prob: 18.75                     Times: 3
             Action: 3          To State: 86      Prob: 6.25                      Times: 1
             Action: 4          To State: 45      Prob: 6.25                      Times: 1
             Action: 5          To State: 45      Prob: 6.25                      Times: 1
             Action: 28         To State: 602     Prob: 6.25                      Times: 1
             Action: 30         To State: 560     Prob: 6.25                      Times: 1
             Action: 30         To State: 87      Prob: 12.5                      Times: 2

State: S45   a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,-c
             Action: 3          To State: 47      Prob: 50                        Times: 2
             Action: 3          To State: 46      Prob: 50                        Times: 2

             a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,-c,b
State: S46   0

State: S47   a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,-c,b
             Action: 9          To State: 48      Prob: 60                        Times: 3
             Action: 10         To State: 6       Prob: 40                        Times: 2

State: S48   a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,-c,b,b*-c 0

State: S49   a>b,c>d,-(a>d),-(-a+d),a*-d,d>a 0

State: S50   a>b,c>d,-(a>d),(a*-d),b,-c 0
```

```
State: S51    a>b,c>d,-(a>d),-(-a+d),--a*d 0

State: S52    a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c 0

State: S53    a>b,c>d,-(a>d),-d>-c
              Action: 8         To State: 754    Prob: 11.7647058823529       Times: 2
              Action: 8         To State: 254    Prob: 47.0588235294118       Times: 8
              Action: 29        To State: 739    Prob: 5.88235294117647       Times: 1
              Action: 29        To State: 645    Prob: 5.88235294117647       Times: 1
              Action: 30        To State: 762    Prob: 5.88235294117647       Times: 1
              Action: 30        To State: 54     Prob: 5.88235294117647       Times: 1
              Action: 31        To State: 673    Prob: 11.7647058823529       Times: 2
              Action: 31        To State: 518    Prob: 5.88235294117647       Times: 1

State: S54    a>b,c>d,-(a>d),-d>-c,a*-d
              Action: 3         To State: 55     Prob: 33.3333333333333       Times: 1
              Action: 8         To State: 647    Prob: 33.3333333333333       Times: 1
              Action: 8         To State: 646    Prob: 33.3333333333333       Times: 1

State: S55    a>b,c>d,-(a>d),-d>-c,a*-d,b
              Action: 30        To State: 56     Prob: 100                    Times: 1

State: S56    a>b,c>d,-(a>d),-d>-c,a*-d,b,-c+d
              Action: 2         To State: 58     Prob: 50                     Times: 1
              Action: 3         To State: 57     Prob: 50                     Times: 1

State: S57    a>b,c>d,-(a>d),-d>-c,a*-d,b,-c+d,-d 0

State: S58    a>b,c>d,-(a>d),-d>-c,a*-d,b,-c+d,-c
              Action: 10        To State: 6      Prob: 100                    Times: 1

State: S59    a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,(a+c)>(b+d)
              Action: 3         To State: 60     Prob: 50                     Times: 1
              Action: 9         To State: 61     Prob: 50                     Times: 1

State: S60    a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,(a+c)>(b+d),b+d 0

State: S61    a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,(a+c)>(b+d),a+c
              Action: 3         To State: 62     Prob: 100                    Times: 1

State: S62    a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,(a+c)>(b+d),a+c,b+d
              Action: 4         To State: 63     Prob: 100                    Times: 1

State: S63    a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,(a+c)>(b+d),a+c,b+d,-c
              Action: 10        To State: 6      Prob: 100                    Times: 1

State: S64    a>b,c>d,-(a>d),-(-a+d),-a+b
              Action: 2         To State: 598    Prob: 33.3333333333333       Times: 1
              Action: 27        To State: 65     Prob: 33.3333333333333       Times: 1
```

```
               Action: 30        To State: 102    Prob: 33.3333333333333        Times: 1

State: S65     a>b,c>d,-(a>d),-(-a+d),-a+b,a*-d
               Action: 30        To State: 66     Prob: 100                     Times: 1

State: S66     a>b,c>d,-(a>d),-(-a+d),-a+b,a*-d,-c+d
               Action: 2         To State: 68     Prob: 33.3333333333333        Times: 1
               Action: 2         To State: 67     Prob: 66.6666666666667        Times: 2

State: S67     a>b,c>d,-(a>d),-(-a+d),-a+b,a*-d,-c+d,-c 0

State: S68     a>b,c>d,-(a>d),-(-a+d),-a+b,a*-d,-c+d,-c
               Action: 2         To State: 70     Prob: 50                      Times: 1
               Action: 2         To State: 69     Prob: 50                      Times: 1

State: S69     a>b,c>d,-(a>d),-(-a+d),-a+b,a*-d,-c+d,-c,b 0

State: S70     a>b,c>d,-(a>d),-(-a+d),-a+b,a*-d,-c+d,-c,b
               Action: 10        To State: 6      Prob: 100                     Times: 1

State: S71     a>b,c>d,-(a>d),-(-a+d),a*d 0

State: S72     a>b,c>d,-(a>d),(a*c)>(b*d)
               Action: 1         To State: 656    Prob: 18.1818181818182        Times: 2
               Action: 30        To State: 757    Prob: 9.09090909090909        Times: 1
               Action: 30        To State: 447    Prob: 9.09090909090909        Times: 1
               Action: 30        To State: 73     Prob: 36.3636363636364        Times: 4
               Action: 31        To State: 772    Prob: 9.09090909090909        Times: 1
               Action: 31        To State: 771    Prob: 9.09090909090909        Times: 1
               Action: 31        To State: 73     Prob: 9.09090909090909        Times: 1

State: S73     a>b,c>d,-(a>d),(a*c)>(b*d),-(-a+d)
               Action: 27        To State: 74     Prob: 60                      Times: 3
               Action: 28        To State: 74     Prob: 40                      Times: 2

               a>b,c>d,-(a>d),(a*c)>(b*d),-(-a+d),a*-
State: S74     d
               Action: 3         To State: 632    Prob: 12.5                    Times: 1
               Action: 3         To State: 486    Prob: 37.5                    Times: 3
               Action: 3         To State: 485    Prob: 12.5                    Times: 1
               Action: 5         To State: 485    Prob: 12.5                    Times: 1
               Action: 30        To State: 676    Prob: 12.5                    Times: 1
               Action: 30        To State: 75     Prob: 12.5                    Times: 1

State: S75     a>b,c>d,-(a>d),(a*c)>(b*d),-(-a+d),a*-d,-c+d
               Action: 3         To State: 77     Prob: 25                      Times: 1
               Action: 27        To State: 76     Prob: 50                      Times: 2
               Action: 28        To State: 76     Prob: 25                      Times: 1

State: S76     a>b,c>d,-(a>d),(a*c)>(b*d),-(-a+d),a*-d,-c+d,c*-d 0
```

```
State: S77    a>b,c>d,-(a>d),(a*c)>(b*d),-(-a+d),a*-d,-c+d,b
              Action: 29        To State: 78      Prob: 100                     Times: 1

State: S78    a>b,c>d,-(a>d),(a*c)>(b*d),-(-a+d),a*-d,-c+d,b,-d>-c
              Action: 30        To State: 79      Prob: 100                     Times: 1

State: S79    a>b,c>d,-(a>d),(a*c)>(b*d),-(-a+d),a*-d,-c+d,b,-d>-c,-c+d
              Action: 3         To State: 80      Prob: 100                     Times: 1

State: S80    a>b,c>d,-(a>d),(a*c)>(b*d),-(-a+d),a*-d,-c+d,b,-d>-c,-c+d,-c
              Action: 10        To State: 6       Prob: 100                     Times: 1

State: S81    a>b,c>d,-(a>d),-a*d 0

State: S82    a>b,c>d,-(a>d),(a*-d),-c
              Action: 3         To State: 84      Prob: 66.6666666666667        Times: 6
              Action: 3         To State: 83      Prob: 11.1111111111111        Times: 1
              Action: 29        To State: 584     Prob: 11.1111111111111        Times: 1
              Action: 31        To State: 342     Prob: 11.1111111111111        Times: 1

State: S83    a>b,c>d,-(a>d),(a*-d),-c,b 0

State: S84    a>b,c>d,-(a>d),(a*-d),-c,b
              Action: 10        To State: 6       Prob: 100                     Times: 6

State: S85    a>b,c>d,-(a>d),a+-b 0

              a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,c*-d
State: S86    0

State: S87    a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,-c+d
              Action: 3         To State: 88      Prob: 33.3333333333333        Times: 1
              Action: 9         To State: 410     Prob: 33.3333333333333        Times: 1
              Action: 10        To State: 6       Prob: 33.3333333333333        Times: 1

State: S88    a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,-c+d,b
              Action: 2         To State: 89      Prob: 100                     Times: 1

State: S89    a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,-c+d,b,-c
              Action: 10        To State: 6       Prob: 100                     Times: 1

State: S90    a>b,c>d,-(a>d),-(-d>-a),-(d+-a),a*-d,-d>-c,-c 0

State: S91    a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c
              Action: 1         To State: 679     Prob: 6.25                    Times: 1
              Action: 1         To State: 609     Prob: 6.25                    Times: 1
              Action: 3         To State: 530     Prob: 18.75                   Times: 3
              Action: 3         To State: 253     Prob: 18.75                   Times: 3
              Action: 3         To State: 202     Prob: 25                      Times: 4
              Action: 8         To State: 800     Prob: 12.5                    Times: 2
              Action: 8         To State: 92      Prob: 6.25                    Times: 1
```

```
                Action: 29              To State: 476      Prob: 6.25                        Times: 1

State: S92      a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,(a*-d)>(b*-c)
                Action: 3               To State: 6        Prob: 100                         Times: 1

State: S93      a>b,c>d,-(a>d),(a*-d),(a*c)>(b*d) 0

State: S94      a>b,c>d,-(a>d),(a*-d),-(a*-b)
                Action: 28              To State: 96       Prob: 66.6666666666667            Times: 4
                Action: 28              To State: 95       Prob: 33.3333333333333            Times: 2

State: S95      a>b,c>d,-(a>d),(a*-d),-(a*-b),-a+b 0

State: S96      a>b,c>d,-(a>d),(a*-d),-(a*-b),-a+b
                Action: 9               To State: 112      Prob: 25                          Times: 1
                Action: 28              To State: 97       Prob: 25                          Times: 1
                Action: 30              To State: 383      Prob: 25                          Times: 1
                Action: 31              To State: 97       Prob: 25                          Times: 1

State: S97      a>b,c>d,-(a>d),(a*-d),-(a*-b),-a+b,-(c*-d)
                Action: 28              To State: 98       Prob: 100                         Times: 2

State: S98      a>b,c>d,-(a>d),(a*-d),-(a*-b),-a+b,-(c*-d),-c+d
                Action: 2               To State: 100      Prob: 25                          Times: 1
                Action: 2               To State: 99       Prob: 75                          Times: 3

State: S99      a>b,c>d,-(a>d),(a*-d),-(a*-b),-a+b,-(c*-d),-c+d,-c 0

State:
S100            a>b,c>d,-(a>d),(a*-d),-(a*-b),-a+b,-(c*-d),-c+d,-c
                Action: 2               To State: 101      Prob: 100                         Times: 1

State:
S101            a>b,c>d,-(a>d),(a*-d),-(a*-b),-a+b,-(c*-d),-c+d,-c,b
                Action: 10              To State: 6        Prob: 100                         Times: 1

State:
S102            a>b,c>d,-(a>d),-(-a+d),-a+b,-c+d
                Action: 27              To State: 103      Prob: 100                         Times: 1

State:
S103            a>b,c>d,-(a>d),-(-a+d),-a+b,-c+d,a*-d
                Action: 2               To State: 106      Prob: 20                          Times: 1
                Action: 2               To State: 104      Prob: 40                          Times: 2
                Action: 5               To State: 105      Prob: 20                          Times: 1
                Action: 7               To State: 104      Prob: 20                          Times: 1

State:
S104            a>b,c>d,-(a>d),-(-a+d),-a+b,-c+d,a*-d,b 0

State:
S105            a>b,c>d,-(a>d),-(-a+d),-a+b,-c+d,a*-d,a 0
```

```
State:
S106        a>b,c>d,-(a>d),-(-a+d),-a+b,-c+d,a*-d,b
            Action: 2          To State: 107    Prob: 100                      Times: 1

State:
S107        a>b,c>d,-(a>d),-(-a+d),-a+b,-c+d,a*-d,b,-c

State:
S108        a>b,c>d,-(a>d),-a=-d 0

State:
S109        a>b,c>d,-(a>d),(a*-d),b*-d 0

State:
S110        a>b,c>d,-(a>d),(a*-d),b 0

State:
S111        a>b,c>d,-(a>d),(a*-d),-b*a 0

State:
S112        a>b,c>d,-(a>d),(a*-d),-(a*-b),-a+b,(-a+b)+a

State:
S113        a>b,c>d,-(a>d),a>d 0

State:      a>b,c>d,-(a>d),-d
S114        0

State:      a>b,c>d,-(a>d),a*-
S115        d
            Action: 1          To State: 572    Prob: 1.61290322580645         Times: 1
            Action: 1          To State: 145    Prob: 1.61290322580645         Times: 1
            Action: 1          To State: 144    Prob: 1.61290322580645         Times: 1
                                                                               Times:
            Action: 3          To State: 272    Prob: 17.741935483871          11
                                                                               Times:
            Action: 3          To State: 116    Prob: 33.8709677419355         21
            Action: 4          To State: 412    Prob: 8.06451612903226         Times: 5
            Action: 4          To State: 272    Prob: 1.61290322580645         Times: 1
            Action: 5          To State: 272    Prob: 1.61290322580645         Times: 1
            Action: 8          To State: 808    Prob: 1.61290322580645         Times: 1
            Action: 10         To State: 326    Prob: 1.61290322580645         Times: 1
            Action: 29         To State: 346    Prob: 3.2258064516129          Times: 2
            Action: 29         To State: 181    Prob: 1.61290322580645         Times: 1
            Action: 29         To State: 158    Prob: 11.2903225806452         Times: 7
            Action: 29         To State: 157    Prob: 1.61290322580645         Times: 1
            Action: 30         To State: 532    Prob: 1.61290322580645         Times: 1
            Action: 30         To State: 132    Prob: 3.2258064516129          Times: 2
            Action: 31         To State: 432    Prob: 1.61290322580645         Times: 1
            Action: 31         To State: 431    Prob: 1.61290322580645         Times: 1
            Action: 32         To State: 430    Prob: 3.2258064516129          Times: 2
```

```
State:
S116        a>b,c>d,-(a>d),a*-d,b
            Action: 3       To State: 273     Prob: 14.2857142857143      Times: 4
            Action: 4       To State: 273     Prob: 10.7142857142857      Times: 3
                                                                          Times:
            Action: 4       To State: 117     Prob: 60.7142857142857      17
            Action: 29      To State: 659     Prob: 3.57142857142857      Times: 1
            Action: 30      To State: 314     Prob: 10.7142857142857      Times: 3

State:
S117        a>b,c>d,-(a>d),a*-d,b,-c
            Action: 10      To State: 759     Prob: 10                    Times: 2
                                                                          Times:
            Action: 10      To State: 6       Prob: 90                    18

State:
S118        a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c>-d 0

State:
S119        a>b,c>d,-(a>d),-(-a+d),a*-d,b,-d>-c
            Action: 1       To State: 120     Prob: 4.76190476190476      Times: 1
                                                                          Times:
            Action: 3       To State: 120     Prob: 85.7142857142857      18
            Action: 9       To State: 339     Prob: 4.76190476190476      Times: 1
            Action: 30      To State: 828     Prob: 4.76190476190476      Times: 1

State:
S120        a>b,c>d,-(a>d),-(-a+d),a*-d,b,-d>-c,-c
            Action: 1       To State: 6       Prob: 4.76190476190476      Times: 1
            Action: 3       To State: 121     Prob: 4.76190476190476      Times: 1
            Action: 9       To State: 121     Prob: 9.52380952380952      Times: 2
                                                                          Times:
            Action: 10      To State: 6       Prob: 80.9523809523809      17

State:
S121        a>b,c>d,-(a>d),-(-a+d),a*-d,b,-d>-c,-c,b*-c 0

State:
S122        a>b,c>d,-(a>d),-a+b 0

State:
S123        a>b,c>d,-(a>d),-a+b,-c+b 0

State:
S124        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,-c 0

State:
S125        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,d 0

State:
S126        a>b,c>d,-(a>d),-(-a+d),a 0

State:      a>b,c>d,-(a>d),-(-a+d),a+-d 0
```

```
S127

State:
S128        a>b,c>d,-(a>d),-(-a+b) 0

State:      a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*d
S129        0

State:
S130        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b*-d 0

State:
S131        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b*-d
            Action: 7        To State: 6      Prob: 100                    Times: 1

State:
S132        a>b,c>d,-(a>d),a*-d,-c+d
            Action: 2        To State: 134    Prob: 25                     Times: 1
            Action: 2        To State: 133    Prob: 50                     Times: 2
            Action: 30       To State: 355    Prob: 25                     Times: 1

State:
S133        a>b,c>d,-(a>d),a*-d,-c+d,-c 0

State:
S134        a>b,c>d,-(a>d),a*-d,-c+d,-c
            Action: 30       To State: 135    Prob: 100                    Times: 1

State:
S135        a>b,c>d,-(a>d),a*-d,-c+d,-c,-a+b
            Action: 2        To State: 136    Prob: 100                    Times: 1

State:
S136        a>b,c>d,-(a>d),a*-d,-c+d,-c,-a+b,b
            Action: 10       To State: 6      Prob: 100                    Times: 1

State:
S137        a>b,c>d,-(a>d),-a+b,-c+d 0

State:
S138        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,-c
            Action: 2        To State: 197    Prob: 16.6666666666667       Times: 2
            Action: 2        To State: 139    Prob: 41.6666666666667       Times: 5
            Action: 3        To State: 139    Prob: 41.6666666666667       Times: 5

State:
S139        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,-c,b
                                                                           Times:
            Action: 10       To State: 6      Prob: 100                    10

State:
S140        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d
            Action: 2        To State: 649    Prob: 5                      Times: 1
```

```
                Action: 2          To State: 214     Prob: 30                   Times: 6
                Action: 2          To State: 141     Prob: 30                   Times: 6
                Action: 3          To State: 650     Prob: 5                    Times: 1
                Action: 9          To State: 566     Prob: 5                    Times: 1
                Action: 29         To State: 695     Prob: 5                    Times: 1
                Action: 30         To State: 822     Prob: 5                    Times: 1
                Action: 30         To State: 538     Prob: 5                    Times: 1
                Action: 30         To State: 414     Prob: 10                   Times: 2

State:
S141            a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-c
                Action: 3          To State: 142     Prob: 33.3333333333333     Times: 2
                Action: 30         To State: 224     Prob: 66.6666666666667     Times: 4

State:
S142            a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-c,b
                Action: 10         To State: 215     Prob: 33.3333333333333     Times: 1
                Action: 10         To State: 6       Prob: 66.6666666666667     Times: 2

State:
S143            a>b,c>d,-(a>d),-(a*-d) 0

State:
S144            a>b,c>d,-(a>d),a*-d,-(c>b) 0

State:
S145            a>b,c>d,-(a>d),a*-d,-(c>b)
                Action: 29         To State: 147     Prob: 33.3333333333333     Times: 1
                Action: 29         To State: 146     Prob: 66.6666666666667     Times: 2

State:
S146            a>b,c>d,-(a>d),a*-d,-(c>b),-b>-c 0

State:
S147            a>b,c>d,-(a>d),a*-d,-(c>b),-(-b>-c)

State:
S148            a>b,c>d,-(a>d),a+b 0

State:
S149            a>b,c>d,-(a>d),-(-a>d) 0

State:
S150            a>b,c>d,-(a>d),(a+c)>(b+d)
                Action: 2          To State: 589     Prob: 25                   Times: 2
                Action: 28         To State: 590     Prob: 12.5                 Times: 1
                Action: 30         To State: 661     Prob: 12.5                 Times: 1
                Action: 30         To State: 592     Prob: 12.5                 Times: 1
                Action: 30         To State: 316     Prob: 12.5                 Times: 1
                Action: 31         To State: 591     Prob: 12.5                 Times: 1
                Action: 31         To State: 151     Prob: 12.5                 Times: 1
```

```
State:
S151        a>b,c>d,-(a>d),(a+c)>(b+d),a*-d
            Action: 3          To State: 152     Prob: 100                         Times: 1

State:
S152        a>b,c>d,-(a>d),(a+c)>(b+d),a*-d,b
            Action: 4          To State: 153     Prob: 100                         Times: 1

State:
S153        a>b,c>d,-(a>d),(a+c)>(b+d),a*-d,b,-c
            Action: 10         To State: 6       Prob: 100                         Times: 1

State:
S154        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,c+-d 0

State:
S155        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,c*d 0

State:
S156        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,a>d 0

State:
S157        a>b,c>d,-(a>d),a*-d,-d> 0

State:
S158        a>b,c>d,-(a>d),a*-d,-d>-c
            Action: 1          To State: 680     Prob: 4.76190476190476            Times: 1
            Action: 2          To State: 159     Prob: 4.76190476190476            Times: 1
            Action: 3          To State: 681     Prob: 4.76190476190476            Times: 1
            Action: 3          To State: 160     Prob: 23.8095238095238            Times: 5
                                                                                   Times:
            Action: 3          To State: 159     Prob: 61.9047619047619            13

State:
S159        a>b,c>d,-(a>d),a*-d,-d>-c,-c 0

State:
S160        a>b,c>d,-(a>d),a*-d,-d>-c,-c
            Action: 3          To State: 162     Prob: 80                          Times: 4
            Action: 3          To State: 161     Prob: 20                          Times: 1

State:
S161        a>b,c>d,-(a>d),a*-d,-d>-c,-c,b 0

State:
S162        a>b,c>d,-(a>d),a*-d,-d>-c,-c,b
            Action: 10         To State: 6       Prob: 100                         Times: 4

State:
S163        a>b,c>d,-(a>d),-(-a+d),a*-d,a>-(b*d) 0

State:      a>b,c>d,-(a>d),-(-a+d),a*-
S164        d,(a*c)>(b*d)
```

```
                Action: 4           To State: 165     Prob: 25                        Times: 1
                Action: 30          To State: 505     Prob: 75                        Times: 3

State:
S165            a>b,c>d,-(a>d),-(-a+d),a*-d,(a*c)>(b*d),-c
                Action: 3           To State: 166     Prob: 100                       Times: 1

State:
S166            a>b,c>d,-(a>d),-(-a+d),a*-d,(a*c)>(b*d),-c,b
                Action: 10          To State: 6       Prob: 100                       Times: 1

State:
S167            a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,a

State:
S168            a>b,c>d,-(a>d),-a+b,-c+d,-d>-a 0

State:
S169            a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,a*-b 0

State:
S170            a>b,c>d,-(a>d),-(-a+d),a*-d,-c 0

State:
S171            a>b,c>d,-(a>d),-(-a+d),a*-d,-c
                                                                                      Times:
                Action: 3           To State: 172     Prob: 75                        24
                Action: 5           To State: 341     Prob: 3.125                     Times: 1
                Action: 10          To State: 6       Prob: 6.25                      Times: 2
                Action: 10          To State: 604     Prob: 3.125                     Times: 1
                Action: 10          To State: 188     Prob: 6.25                      Times: 2
                Action: 30          To State: 185     Prob: 6.25                      Times: 2

State:
S172            a>b,c>d,-(a>d),-(-a+d),a*-d,-c,b
                Action: 9           To State: 304     Prob: 6.25                      Times: 2
                Action: 10          To State: 304     Prob: 15.625                    Times: 5
                Action: 10          To State: 219     Prob: 6.25                      Times: 2
                                                                                      Times:
                Action: 10          To State: 6       Prob: 71.875                    23

State:
S173            a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b
                Action: 2           To State: 482     Prob: 16.6666666666667          Times: 3
                Action: 29          To State: 237     Prob: 5.55555555555556          Times: 1
                                                                                      Times:
                Action: 30          To State: 175     Prob: 61.1111111111111          11
                Action: 30          To State: 174     Prob: 16.6666666666667          Times: 3

State:
S174            a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d 0

State:          a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d
```

```
S175
             Action: 2      To State: 265    Prob: 47.3684210526316      Times: 9
             Action: 2      To State: 176    Prob: 5.26315789473684      Times: 1
             Action: 3      To State: 177    Prob: 10.5263157894737      Times: 2
             Action: 7      To State: 686    Prob: 5.26315789473684      Times: 1
             Action: 7      To State: 264    Prob: 5.26315789473684      Times: 1
             Action: 7      To State: 204    Prob: 5.26315789473684      Times: 1
             Action: 10     To State: 687    Prob: 5.26315789473684      Times: 1
             Action: 10     To State: 633    Prob: 5.26315789473684      Times: 1
             Action: 10     To State: 229    Prob: 5.26315789473684      Times: 1
             Action: 29     To State: 230    Prob: 5.26315789473684      Times: 1

State:
S176         a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,b 0

State:
S177         a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,b
             Action: 2      To State: 179    Prob: 37.5                  Times: 3
             Action: 2      To State: 178    Prob: 50                    Times: 4
             Action: 4      To State: 179    Prob: 12.5                  Times: 1

State:
S178         a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,b,-c 0

State:
S179         a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,b,-c
             Action: 10     To State: 6      Prob: 100                   Times: 4

State:
S180         a>b,c>d,-(a>d),-a>-d 0

State:
S181         a>b,c>d,-(a>d),a*-d,-d=-c 0

State:
S182         a>b,c>d,-(a>d),-(-a+d),a*-d,-d 0

State:
S183         a>b,c>d,-(a>d),-(-a+d),a*-d,b*-c 0

State:
S184         a>b,c>d,-(a>d),-c+d,-a+b
             Action: 10     To State: 6      Prob: 50                    Times: 1
             Action: 31     To State: 282    Prob: 50                    Times: 1

State:
S185         a>b,c>d,-(a>d),-(-a+d),a*-d,-c,-a+b
             Action: 2      To State: 187    Prob: 66.6666666666667      Times: 2
             Action: 2      To State: 186    Prob: 33.3333333333333      Times: 1

State:       a>b,c>d,-(a>d),-(-a+d),a*-d,-c,-a+b,b
S186         0
```

```
State:
S187        a>b,c>d,-(a>d),-(-a+d),a*-d,-c,-a+b,b
            Action: 10          To State: 6         Prob: 100                       Times: 2

State:
S188        a>b,c>d,-(a>d),-(-a+d),a*-d,-c,a*-c
            Action: 30          To State: 189       Prob: 100                       Times: 1

State:
S189        a>b,c>d,-(a>d),-(-a+d),a*-d,-c,a*-c,-a+b
            Action: 2           To State: 190       Prob: 100                       Times: 1

State:
S190        a>b,c>d,-(a>d),-(-a+d),a*-d,-c,a*-c,-a+b,b
            Action: 10          To State: 6         Prob: 100                       Times: 1

State:
S191        a>b,c>d,-(a>d),-c+d 0

State:
S192        a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d,-d 0

State:
S193        a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d,-c
            Action: 30          To State: 194       Prob: 100                       Times: 1

State:
S194        a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d,-c,-a+b
            Action: 2           To State: 196       Prob: 33.3333333333333          Times: 1
            Action: 2           To State: 195       Prob: 66.6666666666667          Times: 2

State:
S195        a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d,-c,-a+b,b 0

State:
S196        a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d,-c,-a+b,b
            Action: 10          To State: 6         Prob: 100                       Times: 1

State:
S197        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,-c,b 0

State:
S198        a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c+d
            Action: 2           To State: 200       Prob: 50                        Times: 6
            Action: 4           To State: 200       Prob: 16.6666666666667          Times: 2
            Action: 4           To State: 199       Prob: 8.33333333333333          Times: 1
            Action: 29          To State: 575       Prob: 8.33333333333333          Times: 1
            Action: 29          To State: 574       Prob: 8.33333333333333          Times: 1
            Action: 30          To State: 820       Prob: 8.33333333333333          Times: 1

State:
S199        a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c+d,-c 0
```

```
State:
S200       a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c+d,-c
           Action: 9          To State: 201     Prob: 12.5                 Times: 1
           Action: 10         To State: 830     Prob: 12.5                 Times: 1
           Action: 10         To State: 6       Prob: 75                   Times: 6

State:
S201       a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c+d,-c,b+-c 0

State:
S202       a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,-c
           Action: 3          To State: 203     Prob: 75                   Times: 3
           Action: 30         To State: 444     Prob: 25                   Times: 1

State:
S203       a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,-c,b
           Action: 10         To State: 6       Prob: 100                  Times: 3

State:
S204       a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,(a+c)>(b+d)
           Action: 30         To State: 205     Prob: 100                  Times: 1

State:
S205       a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,(a+c)>(b+d),-(a+c)+(b+d)
           Action: 8          To State: 208     Prob: 25                   Times: 1
           Action: 8          To State: 207     Prob: 25                   Times: 1
           Action: 27         To State: 206     Prob: 50                   Times: 2

State:
S206       a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,(a+c)>(b+d),-(a+c)+(b+d),(-a*-c)+(b+d) 0

State:
S207       a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,(a+c)>(b+d),-(a+c)+(b+d),(a*c)>(b*d) 0

State:
S208       a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,(a+c)>(b+d),-(a+c)+(b+d),(a*c)>(b*d)
           Action: 30         To State: 209     Prob: 100                  Times: 1

State:     a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,(a+c)>(b+d),-(a+c)+(b+d),(a*c)>(b*d),-
S209       (a*c)+(b*d)

State:
S210       a>b,c>d,-(a>d),-(a*-b)
           Action: 28         To State: 211     Prob: 36.3636363636364     Times: 4
           Action: 30         To State: 211     Prob: 9.09090909090909     Times: 1
           Action: 31         To State: 220     Prob: 54.5454545454545     Times: 6

State:
S211       a>b,c>d,-(a>d),-(a*-b),-a+b
           Action: 29         To State: 212     Prob: 20                   Times: 1
           Action: 30         To State: 763     Prob: 20                   Times: 1
           Action: 31         To State: 241     Prob: 60                   Times: 3
```

```
State:
S212        a>b,c>d,-(a>d),-(a*-b),-a+b,-b>-a

State:
S213        a>b,c>d,-(a>d),-a+b,-c+d,a*-d
            Action: 2      To State: 327    Prob: 57.1428571428571     Times: 8
            Action: 3      To State: 328    Prob: 7.14285714285714     Times: 1
            Action: 4      To State: 606    Prob: 14.2857142857143     Times: 2
            Action: 4      To State: 327    Prob: 7.14285714285714     Times: 1
            Action: 10     To State: 608    Prob: 7.14285714285714     Times: 1
            Action: 10     To State: 607    Prob: 7.14285714285714     Times: 1

State:
S214        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-c 0

State:
S215        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-c,b,b*-c 0

State:
S216        a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d,b
            Action: 2      To State: 218    Prob: 40                   Times: 4
            Action: 2      To State: 217    Prob: 60                   Times: 6

State:
S217        a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d,b,-c 0

State:
S218        a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d,b,-c
            Action: 3      To State: 565    Prob: 20                   Times: 1
            Action: 10     To State: 6      Prob: 80                   Times: 4

State:
S219        a>b,c>d,-(a>d),-(-a+d),a*-d,-c,b,b+-c 0

State:
S220        a>b,c>d,-(a>d),-(a*-b),-(c*-d)
            Action: 28     To State: 348    Prob: 50                   Times: 4
            Action: 28     To State: 347    Prob: 12.5                 Times: 1
            Action: 29     To State: 367    Prob: 12.5                 Times: 1
            Action: 31     To State: 221    Prob: 25                   Times: 2

State:
S221        a>b,c>d,-(a>d),-(a*-b),-(c*-d),a*-d
            Action: 3      To State: 222    Prob: 100                  Times: 2

State:
S222        a>b,c>d,-(a>d),-(a*-b),-(c*-d),a*-d,b
            Action: 4      To State: 223    Prob: 100                  Times: 2

State:
S223        a>b,c>d,-(a>d),-(a*-b),-(c*-d),a*-d,b,-c
            Action: 10     To State: 6      Prob: 100                  Times: 2
```

```
State:
S224        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-c,-a+b
            Action: 2       To State: 340     Prob: 37.5                   Times: 3
            Action: 2       To State: 225     Prob: 37.5                   Times: 3
            Action: 3       To State: 225     Prob: 12.5                   Times: 1
            Action: 5       To State: 567     Prob: 12.5                   Times: 1

State:
S225        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-c,-a+b,b
            Action: 3       To State: 568     Prob: 20                     Times: 1
            Action: 10      To State: 6       Prob: 80                     Times: 4

State:
S226        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,-(b>d) 0

State:
S227        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,-(a>c) 0

State:
S228        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,-(c>a) 0

State:
S229        a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,(a*-d)+(-a*b) 0

State:
S230        a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,-d>-c
            Action: 3       To State: 232     Prob: 33.3333333333333       Times: 1
            Action: 30      To State: 231     Prob: 66.6666666666667       Times: 2

State:
S231        a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,-d>-c,-a>-d 0

State:
S232        a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,-d>-c,b
            Action: 3       To State: 234     Prob: 50                     Times: 1
            Action: 3       To State: 233     Prob: 50                     Times: 1

State:
S233        a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,-d>-c,b,-d 0

State:
S234        a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,-d>-c,b,-c
            Action: 10      To State: 235     Prob: 100                    Times: 1

State:
S235        a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,-d>-c,b,-c,b*-c 0

State:
S236        a>b,c>d,-(a>d),-a+d 0

State:
S237        a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-d>-c
```

```
                Action: 30          To State: 238     Prob: 100                    Times: 1

State:
S238            a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-d>-c,-c+d
                Action: 10          To State: 6       Prob: 100                    Times: 1

State:
S239            a>b,c>d,-(a>d),-(a+d) 0

State:
S240            a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,-c 0

State:
S241            a>b,c>d,-(a>d),-(a*-b),-a+b,-(c*-d)
                Action: 28          To State: 242     Prob: 50                     Times: 2
                Action: 30          To State: 242     Prob: 25                     Times: 1
                Action: 31          To State: 495     Prob: 25                     Times: 1

State:          a>b,c>d,-(a>d),-(a*-b),-a+b,-(c*-d),-
S242            c+d
                Action: 7           To State: 243     Prob: 25                     Times: 1
                Action: 27          To State: 496     Prob: 25                     Times: 1
                Action: 30          To State: 497     Prob: 25                     Times: 1
                Action: 31          To State: 455     Prob: 25                     Times: 1

State:
S243            a>b,c>d,-(a>d),-(a*-b),-a+b,-(c*-d),-c+d,(a+c)>(b+d)
                Action: 31          To State: 244     Prob: 100                    Times: 1

State:
S244            a>b,c>d,-(a>d),-(a*-b),-a+b,-(c*-d),-c+d,(a+c)>(b+d),-(a+c)+(b+d)
                Action: 2           To State: 245     Prob: 100                    Times: 1

State:
S245            a>b,c>d,-(a>d),-(a*-b),-a+b,-(c*-d),-c+d,(a+c)>(b+d),-(a+c)+(b+d),b+d 0

State:
S246            a>b,c>d,-(a>d),-b>-a
                Action: 5           To State: 671     Prob: 10                     Times: 1
                Action: 29          To State: 671     Prob: 10                     Times: 1
                Action: 29          To State: 305     Prob: 30                     Times: 3
                Action: 30          To State: 786     Prob: 10                     Times: 1
                Action: 30          To State: 551     Prob: 30                     Times: 3
                Action: 30          To State: 247     Prob: 10                     Times: 1

State:
S247            a>b,c>d,-(a>d),-b>-a,a*-d
                Action: 3           To State: 249     Prob: 33.3333333333333       Times: 1
                Action: 3           To State: 248     Prob: 66.6666666666667       Times: 2

State:
S248            a>b,c>d,-(a>d),-b>-a,a*-d,b 0
```

```
State:
S249        a>b,c>d,-(a>d),-b>-a,a*-d,b
            Action: 4         To State: 251    Prob: 33.3333333333333       Times: 1
            Action: 4         To State: 250    Prob: 66.6666666666667       Times: 2

State:
S250        a>b,c>d,-(a>d),-b>-a,a*-d,b,-c 0

State:
S251        a>b,c>d,-(a>d),-b>-a,a*-d,b,-c
            Action: 10        To State: 6      Prob: 100                    Times: 1

State:
S252        a>b,c>d,-(a>d),d>a 0

State:
S253        a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,b 0

State:
S254        a>b,c>d,-(a>d),-d>-c,(a*-d)>(b*-c)
            Action: 30        To State: 755    Prob: 12.5                   Times: 1
            Action: 30        To State: 255    Prob: 37.5                   Times: 3
            Action: 31        To State: 255    Prob: 50                     Times: 4

State:      a>b,c>d,-(a>d),-d>-c,(a*-d)>(b*-c),a*-
S255        d
            Action: 3         To State: 6      Prob: 87.5                   Times: 7
            Action: 3         To State: 256    Prob: 12.5                   Times: 1

State:
S256        a>b,c>d,-(a>d),-d>-c,(a*-d)>(b*-c),a*-d,b*-c 0

State:
S257        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,(a*c)>(b>d) 0

State:
S258        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,(a*c)>(b*d)
            Action: 30        To State: 259    Prob: 100                    Times: 1

State:
S259        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,(a*c)>(b*d),-(a*c)+(b*d)
            Action: 28        To State: 260    Prob: 100                    Times: 1

State:
S260        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,(a*c)>(b*d),-(a*c)+(b*d),-a+-c+(b*d)
            Action: 2         To State: 262    Prob: 50                     Times: 1
            Action: 28        To State: 261    Prob: 50                     Times: 1

State:
S261        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,(a*c)>(b*d),-(a*c)+(b*d),-a+-c+(b*d),c*-d 0

State:      a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,(a*c)>(b*d),-(a*c)+(b*d),-a+-c+(b*d),b
```

```
S262
             Action: 2          To State: 263     Prob: 100                        Times: 1

State:
S263         a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,(a*c)>(b*d),-(a*c)+(b*d),-a+-c+(b*d),b,-c
             Action: 10         To State: 6       Prob: 100                        Times: 1

State:
S264         a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,-d 0

State:
S265         a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,-c 0

State:
S266         a>b,c>d,-(a>d),-(-a+d),b
             Action: 4          To State: 267     Prob: 100                        Times: 2

State:
S267         a>b,c>d,-(a>d),-(-a+d),b,-c
             Action: 10         To State: 6       Prob: 100                        Times: 2

State:
S268         a>b,c>d,-(a>d),-(-a+d),a*-d,a>d 0

State:
S269         a>b,c>d,-(a>d),-(-a+d),a*-d,-(a>d)
                                                                                   Times:
             Action: 1          To State: 270     Prob: 91.6666666666667           11
             Action: 3          To State: 271     Prob: 8.33333333333333           Times: 1

State:
S270         a>b,c>d,-(a>d),-(-a+d),a*-d,-(a>d),-(a>c) 0

State:
S271         a>b,c>d,-(a>d),-(-a+d),a*-d,-(a>d),- 0

State:
S272         a>b,c>d,-(a>d),a*-d,b 0

State:
S273         a>b,c>d,-(a>d),a*-d,b,-c 0

State:
S274         a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c,b*-c 0

State:
S275         a>b,c>d,-(a>d),-a+b,a*-d
             Action: 4          To State: 670     Prob: 50                         Times: 1
             Action: 30         To State: 276     Prob: 50                         Times: 1

State:
S276         a>b,c>d,-(a>d),-a+b,a*-d,-c+d
             Action: 2          To State: 278     Prob: 50                         Times: 1
```

```
               Action: 2          To State: 277     Prob: 50                    Times: 1

State:
S277           a>b,c>d,-(a>d),-a+b,a*-d,-c+d,-c 0

State:
S278           a>b,c>d,-(a>d),-a+b,a*-d,-c+d,-c
               Action: 2          To State: 279     Prob: 100                   Times: 1

State:
S279           a>b,c>d,-(a>d),-a+b,a*-d,-c+d,-c,b
               Action: 10         To State: 6       Prob: 100                   Times: 1

State:
S280           a>b,c>d,-(a>d),--(a+-d) 0

State:
S281           a>b,c>d,-(a>d),-d>-a 0

State:
S282           a>b,c>d,-(a>d),-c+d,-a+b,a*-d
               Action: 2          To State: 283     Prob: 100                   Times: 1

State:
S283           a>b,c>d,-(a>d),-c+d,-a+b,a*-d,-c
               Action: 2          To State: 284     Prob: 100                   Times: 1

State:
S284           a>b,c>d,-(a>d),-c+d,-a+b,a*-d,-c,b
               Action: 10         To State: 6       Prob: 100                   Times: 1

State:
S285           a>b,c>d,-(a>d),(a*-d),(a*b)>(c*d) 0

State:
S286           a>b,c>d,-(a>d),(a*-d),(a*c)>(b*d)
               Action: 3          To State: 623     Prob: 25                    Times: 1
               Action: 29         To State: 625     Prob: 25                    Times: 1
               Action: 29         To State: 624     Prob: 25                    Times: 1
               Action: 29         To State: 287     Prob: 25                    Times: 1

State:
S287           a>b,c>d,-(a>d),(a*-d),(a*c)>(b*d),-(b*d)>-(a*c)
               Action: 31         To State: 288     Prob: 100                   Times: 1

State:
S288           a>b,c>d,-(a>d),(a*-d),(a*c)>(b*d),-(b*d)>-(a*c),-(-(b*d)*(a*c))
               Action: 28         To State: 289     Prob: 100                   Times: 1

State:
S289           a>b,c>d,-(a>d),(a*-d),(a*c)>(b*d),-(b*d)>-(a*c),-(-(b*d)*(a*c)),-((-b+-d)*a*c) 0

State:         a>b,c>d,-(a>d),-a+-d 0
```

```
S290

State:
S291      a>b,c>d,-(a>d),-a+b,(a*-d)
          Action: 2       To State: 737    Prob: 33.3333333333333      Times: 1
          Action: 4       To State: 736    Prob: 33.3333333333333      Times: 1
          Action: 28      To State: 292    Prob: 33.3333333333333      Times: 1

State:
S292      a>b,c>d,-(a>d),-a+b,(a*-d),a*-d
          Action: 3       To State: 293    Prob: 100                   Times: 1

State:
S293      a>b,c>d,-(a>d),-a+b,(a*-d),a*-d,b
          Action: 4       To State: 295    Prob: 50                    Times: 1
          Action: 4       To State: 294    Prob: 50                    Times: 1

State:
S294      a>b,c>d,-(a>d),-a+b,(a*-d),a*-d,b,-c 0

State:
S295      a>b,c>d,-(a>d),-a+b,(a*-d),a*-d,b,-c
          Action: 10      To State: 6      Prob: 50                    Times: 1
          Action: 10      To State: 296    Prob: 50                    Times: 1

State:
S296      a>b,c>d,-(a>d),-a+b,(a*-d),a*-d,b,-c,b*-c 0

State:
S297      a>b,c>d,-(a>d),-(-d>-a),-d*a 0

State:
S298      a>b,c>d,-(a>d),-(-d>-a),a*-d
          Action: 2       To State: 299    Prob: 33.3333333333333      Times: 1
          Action: 29      To State: 300    Prob: 66.6666666666667      Times: 2

State:
S299      a>b,c>d,-(a>d),-(-d>-a),a*-d,-d>-c 0

State:
S300      a>b,c>d,-(a>d),-(-d>-a),a*-d,-d>-c
          Action: 30      To State: 301    Prob: 100                   Times: 2

State:    a>b,c>d,-(a>d),-(-d>-a),a*-d,-d>-c,-
S301      c+d
          Action: 10      To State: 411    Prob: 50                    Times: 1
          Action: 10      To State: 6      Prob: 50                    Times: 1

State:
S302      a>b,c>d,-(a>d),-(b>c) 0

State:
S303      a>b,c>d,-(a>d),-(a>d) 0
```

```
State:
S304        a>b,c>d,-(a>d),-(-a+d),a*-d,-c,b,b*-c 0

State:
S305        a>b,c>d,-(a>d),-b>-a,-d>-c
            Action: 27      To State: 306     Prob: 33.3333333333333      Times: 1
            Action: 30      To State: 306     Prob: 33.3333333333333      Times: 1
            Action: 31      To State: 488     Prob: 33.3333333333333      Times: 1

State:
S306        a>b,c>d,-(a>d),-b>-a,-d>-c,-(-a+d)
            Action: 27      To State: 308     Prob: 66.6666666666667      Times: 2
            Action: 29      To State: 307     Prob: 33.3333333333333      Times: 1

State:
S307        a>b,c>d,-(a>d),-b>-a,-d>-c,-(-a+d),-a>-d 0

State:      a>b,c>d,-(a>d),-b>-a,-d>-c,-(-a+d),a*-
S308        d
            Action: 3       To State: 309     Prob: 66.6666666666667      Times: 2
            Action: 30      To State: 404     Prob: 33.3333333333333      Times: 1

State:
S309        a>b,c>d,-(a>d),-b>-a,-d>-c,-(-a+d),a*-d,b
            Action: 4       To State: 310     Prob: 100                   Times: 2

State:
S310        a>b,c>d,-(a>d),-b>-a,-d>-c,-(-a+d),a*-d,b,-c
            Action: 10      To State: 6       Prob: 100                   Times: 2

State:
S311        a>b,c>d,-(a>d),a*-d 0

State:
S312        a>b,c>d,-(a>d),(a*-d),-(-a+d)
            Action: 27      To State: 313     Prob: 50                    Times: 1
            Action: 30      To State: 509     Prob: 50                    Times: 1

State:
S313        a>b,c>d,-(a>d),(a*-d),-(-a+d),a*-d

State:
S314        a>b,c>d,-(a>d),a*-d,b,-c+d
            Action: 2       To State: 752     Prob: 33.3333333333333      Times: 1
            Action: 4       To State: 315     Prob: 66.6666666666667      Times: 2

State:
S315        a>b,c>d,-(a>d),a*-d,b,-c+d,-c
            Action: 10      To State: 6       Prob: 100                   Times: 3

State:
S316        a>b,c>d,-(a>d),(a+c)>(b+d),-(-a+d)
```

```
              Action: 27          To State: 317      Prob: 100                      Times: 1

State:
S317          a>b,c>d,-(a>d),(a+c)>(b+d),-(-a+d),a*-d
              Action: 3           To State: 318      Prob: 100                      Times: 1

State:
S318          a>b,c>d,-(a>d),(a+c)>(b+d),-(-a+d),a*-d,b
              Action: 4           To State: 319      Prob: 100                      Times: 1

State:
S319          a>b,c>d,-(a>d),(a+c)>(b+d),-(-a+d),a*-d,b,-c
              Action: 9           To State: 321      Prob: 50                       Times: 2
              Action: 9           To State: 320      Prob: 25                       Times: 1
              Action: 30          To State: 322      Prob: 25                       Times: 1

State:
S320          a>b,c>d,-(a>d),(a+c)>(b+d),-(-a+d),a*-d,b,-c,b*-c 0

State:
S321          a>b,c>d,-(a>d),(a+c)>(b+d),-(-a+d),a*-d,b,-c,b+-c 0

State:
S322          a>b,c>d,-(a>d),(a+c)>(b+d),-(-a+d),a*-d,b,-c,-a+b
              Action: 27          To State: 323      Prob: 100                      Times: 1

State:
S323          a>b,c>d,-(a>d),(a+c)>(b+d),-(-a+d),a*-d,b,-c,-a+b,-(c+a)+(-c*b)
              Action: 10          To State: 6        Prob: 50                       Times: 1
              Action: 31          To State: 324      Prob: 50                       Times: 1

State:
S324          a>b,c>d,-(a>d),(a+c)>(b+d),-(-a+d),a*-d,b,-c,-a+b,-(c+a)+(-c*b),(c>-a)+(-c*b) 0

State:
S325          a>b,c>d,-(a>d),-(b>d) 0

State:
S326          a>b,c>d,-(a>d),a*-d,a 0

State:
S327          a>b,c>d,-(a>d),-a+b,-c+d,a*-d,b 0

State:
S328          a>b,c>d,-(a>d),-a+b,-c+d,a*-d,b
              Action: 2           To State: 778      Prob: 20                       Times: 1
              Action: 2           To State: 777      Prob: 20                       Times: 1
              Action: 2           To State: 329      Prob: 40                       Times: 2
              Action: 3           To State: 329      Prob: 20                       Times: 1

State:
S329          a>b,c>d,-(a>d),-a+b,-c+d,a*-d,b,-c
              Action: 10          To State: 6        Prob: 100                      Times: 3
```

```
State:
S330          a>b,c>d,-(a>d),(a*c)>(b*d) 0

State:
S331          a>b,c>d,-(a>d),(a*b)>(c*d) 0

State:
S332          a>b,c>d,-(a>d),-(-a+d),a*-d,-(a*-b)
              Action: 8          To State: 333      Prob: 100                    Times: 1

State:
S333          a>b,c>d,-(a>d),-(-a+d),a*-d,-(a*-b),(a*c)>(b*d)
              Action: 31         To State: 334      Prob: 100                    Times: 1

State:
S334          a>b,c>d,-(a>d),-(-a+d),a*-d,-(a*-b),(a*c)>(b*d),-(a*c)+(b*d)
              Action: 28         To State: 335      Prob: 100                    Times: 1

State:
S335          a>b,c>d,-(a>d),-(-a+d),a*-d,-(a*-b),(a*c)>(b*d),-(a*c)+(b*d),(-a+-c)+(b*d)
              Action: 28         To State: 336      Prob: 100                    Times: 1

State:
S336          a>b,c>d,-(a>d),-(-a+d),a*-d,-(a*-b),(a*c)>(b*d),-(a*c)+(b*d),(-a+-c)+(b*d),-a+b
              Action: 2          To State: 337      Prob: 100                    Times: 1

State:
S337          a>b,c>d,-(a>d),-(-a+d),a*-d,-(a*-b),(a*c)>(b*d),-(a*c)+(b*d),(-a+-c)+(b*d),-a+b,b
              Action: 4          To State: 338      Prob: 100                    Times: 1

State:
S338          a>b,c>d,-(a>d),-(-a+d),a*-d,-(a*-b),(a*c)>(b*d),-(a*c)+(b*d),(-a+-c)+(b*d),-a+b,b,-c
              Action: 10         To State: 6        Prob: 100                    Times: 1

State:
S339          a>b,c>d,-(a>d),-(-a+d),a*-d,b,-d>-c,-c 0

State:
S340          a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-c,-a+b,b 0

State:
S341          a>b,c>d,-(a>d),-(-a+d),a*-d,-c,b 0

State:
S342          a>b,c>d,-(a>d),(a*-d),-c,-(a*-b)
              Action: 28         To State: 343      Prob: 100                    Times: 1

State:
S343          a>b,c>d,-(a>d),(a*-d),-c,-(a*-b),-a+b
              Action: 10         To State: 6        Prob: 50                     Times: 1
              Action: 10         To State: 344      Prob: 50                     Times: 1
```

```
State:
S344         a>b,c>d,-(a>d),(a*-d),-c,-(a*-b),-a+b,b*-c 0

State:
S345         a>b,c>d,-(a>d),-a*-d 0

State:
S346         a>b,c>d,-(a>d),a*-d,-c=-d 0

State:
S347         a>b,c>d,-(a>d),-(a*-b),-(c*-d),-a*--b 0

State:
S348         a>b,c>d,-(a>d),-(a*-b),-(c*-d),-a+b
             Action: 28         To State: 349      Prob: 100                          Times: 4

State:       a>b,c>d,-(a>d),-(a*-b),-(c*-d),-a+b,-
S349         c+d
             Action: 3          To State: 388      Prob: 25                           Times: 1
             Action: 30         To State: 350      Prob: 75                           Times: 3

State:
S350         a>b,c>d,-(a>d),-(a*-b),-(c*-d),-a+b,-c+d,-(-a+d)
             Action: 2          To State: 351      Prob: 11.1111111111111             Times: 1
             Action: 5          To State: 351      Prob: 11.1111111111111             Times: 1
             Action: 28         To State: 352      Prob: 11.1111111111111             Times: 1
             Action: 29         To State: 352      Prob: 22.2222222222222             Times: 2
             Action: 29         To State: 351      Prob: 44.4444444444444             Times: 4

State:
S351         a>b,c>d,-(a>d),-(a*-b),-(c*-d),-a+b,-c+d,-(-a+d),a+-d 0

State:
S352         a>b,c>d,-(a>d),-(a*-b),-(c*-d),-a+b,-c+d,-(-a+d),a*-d
             Action: 2          To State: 751      Prob: 80                           Times: 4
             Action: 3          To State: 389      Prob: 20                           Times: 1

State:
S353         a>b,c>d,-(a>d),-a+b,-c+d,-a>-d 0

State:
S354         a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),--a*d 0

State:
S355         a>b,c>d,-(a>d),a*-d,-c+d,-a+b
             Action: 10         To State: 6        Prob: 100                          Times: 1

State:
S356         b*-c,b*-c 0

State:
S357         a>b,c>d,-(a>d),-a+b,-(-a+d),a*d 0
```

```
State:
S358        a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,b
            Action: 4      To State: 360   Prob: 50                  Times: 3
            Action: 4      To State: 359   Prob: 16.6666666666667    Times: 1
            Action: 30     To State: 634   Prob: 33.3333333333333    Times: 2

State:
S359        a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,b,c 0

State:
S360        a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,b,-c
            Action: 10     To State: 776   Prob: 40                  Times: 2
            Action: 10     To State: 6     Prob: 60                  Times: 3

State:
S361        b*-c,(b*-c)>(b*-c) 0

State:
S362        a>b,c>d,-(a>d),-d>-c 0

State:
S363        a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d,b 0

State:
S364        a>b,c>d,-(a>d),(a*-d),a 0

State:
S365        a>b,c>d,-(a>d),(a*-d),b,-d 0

State:
S366        a>b,c>d,-(a>d),(a*-d),b,-c,b*-c 0

State:
S367        a>b,c>d,-(a>d),-(a*-b),-(c*-d),-d>-a 0

State:
S368        a>b,c>d,-(a>d),(a*-d),-d>-c
            Action: 1      To State: 677   Prob: 12.5                Times: 1
            Action: 1      To State: 422   Prob: 12.5                Times: 1
            Action: 3      To State: 424   Prob: 37.5                Times: 3
            Action: 9      To State: 423   Prob: 12.5                Times: 1
            Action: 10     To State: 369   Prob: 12.5                Times: 1
            Action: 30     To State: 370   Prob: 12.5                Times: 1

State:
S369        a>b,c>d,-(a>d),(a*-d),-d>-c,a*-c 0

State:
S370        a>b,c>d,-(a>d),(a*-d),-d>-c,-(-a+d)
            Action: 27     To State: 371   Prob: 100                 Times: 1

State:      a>b,c>d,-(a>d),(a*-d),-d>-c,-(-a+d),a*-
S371        d
```

```
                 Action: 4          To State: 372     Prob: 100              Times: 1

State:
S372             a>b,c>d,-(a>d),(a*-d),-d>-c,-(-a+d),a*-d,-c
                 Action: 30         To State: 373     Prob: 100              Times: 1

State:
S373             a>b,c>d,-(a>d),(a*-d),-d>-c,-(-a+d),a*-d,-c,-a+b
                 Action: 28         To State: 374     Prob: 100              Times: 1

State:
S374             a>b,c>d,-(a>d),(a*-d),-d>-c,-(-a+d),a*-d,-c,-a+b,-(a*-b)
                 Action: 3          To State: 375     Prob: 100              Times: 1

State:
S375             a>b,c>d,-(a>d),(a*-d),-d>-c,-(-a+d),a*-d,-c,-a+b,-(a*-b),b
                 Action: 10         To State: 6       Prob: 100              Times: 1

State:
S376             a>b,c>d,-(a>d),-a+b,-(a*-b) 0

State:
S377             a>b,c>d,-(a>d),-a+b,-(a*-b)
                 Action: 30         To State: 378     Prob: 100              Times: 2

State:
S378             a>b,c>d,-(a>d),-a+b,-(a*-b),-c+d
                 Action: 30         To State: 379     Prob: 50               Times: 1
                 Action: 31         To State: 522     Prob: 50               Times: 1

State:
S379             a>b,c>d,-(a>d),-a+b,-(a*-b),-c+d,-(-a+d)
                 Action: 27         To State: 380     Prob: 100              Times: 1

State:
S380             a>b,c>d,-(a>d),-a+b,-(a*-b),-c+d,-(-a+d),a*-d
                 Action: 3          To State: 381     Prob: 100              Times: 1

State:
S381             a>b,c>d,-(a>d),-a+b,-(a*-b),-c+d,-(-a+d),a*-d,b
                 Action: 4          To State: 382     Prob: 100              Times: 1

State:
S382             a>b,c>d,-(a>d),-a+b,-(a*-b),-c+d,-(-a+d),a*-d,b,-c
                 Action: 10         To State: 6       Prob: 100              Times: 1

State:
S383             a>b,c>d,-(a>d),(a*-d),-(a*-b),-a+b,-c+d
                 Action: 7          To State: 384     Prob: 100              Times: 1

State:
S384             a>b,c>d,-(a>d),(a*-d),-(a*-b),-a+b,-c+d,(a+c)>(b+d)
                 Action: 2          To State: 386     Prob: 50               Times: 1
                 Action: 2              To State: 385      Prob: 50                             Times: 1

State:
S385             a>b,c>d,-(a>d),(a*-d),-(a*-b),-a+b,-c+d,(a+c)>(b+d),b 0

State:
S386             a>b,c>d,-(a>d),(a*-d),-(a*-b),-a+b,-c+d,(a+c)>(b+d),b
                 Action: 2              To State: 387      Prob: 100                            Times: 1

State:
S387             a>b,c>d,-(a>d),(a*-d),-(a*-b),-a+b,-c+d,(a+c)>(b+d),b,-c
                 Action: 10             To State: 6        Prob: 100                            Times: 1

State:
S388             a>b,c>d,-(a>d),-(a*-b),-(c*-d),-a+b,-c+d,a 0

State:
S389             a>b,c>d,-(a>d),-(a*-b),-(c*-d),-a+b,-c+d,-(-a+d),a*-d,b
                 Action: 2              To State: 390      Prob: 100                            Times: 1

State:
S390             a>b,c>d,-(a>d),-(a*-b),-(c*-d),-a+b,-c+d,-(-a+d),a*-d,b,-c
                 Action: 9              To State: 391      Prob: 50                             Times: 1
                 Action: 10             To State: 6        Prob: 50                             Times: 1

State:
S391             a>b,c>d,-(a>d),-(a*-b),-(c*-d),-a+b,-c+d,-(-a+d),a*-d,b,-c,b*-c 0

State:
S392             a>b,c>d,-(a>d),(a*-d),-((a*c)*-(b*d))
                 Action: 28             To State: 394      Prob: 50                             Times: 1
                 Action: 28             To State: 393      Prob: 50                             Times: 1

State:
S393             a>b,c>d,-(a>d),(a*-d),-((a*c)*-(b*d)),-(a*c)+-(b*d) 0

State:
S394             a>b,c>d,-(a>d),(a*-d),-((a*c)*-(b*d)),-(a*c)+(b*d)
                 Action: 28             To State: 395      Prob: 100                            Times: 1

State:
S395             a>b,c>d,-(a>d),(a*-d),-((a*c)*-(b*d)),-(a*c)+(b*d),(-a+-c)+(-b+-d) 0

State:
S396             a>b,c>d,-(a>d),(a*-d),b,-d>-c 0

State:
S397             a>b,c>d,-(a>d),(a*-d),b,-d>-c
                 Action: 3              To State: 398      Prob: 100                            Times: 1

State:
S398             a>b,c>d,-(a>d),(a*-d),b,-d>-c,-c
                 Action: 10             To State: 6        Prob: 100                            Times: 1
```

```
State:
S399        a>b,c>d,-(a>d),-(-a+d),a*-d,b*-d 0

State:
S400        a>b,c>d,-(a>d),-(-a+d),a*-d,b,b*-d
            Action: 29        To State: 401     Prob: 100                  Times: 3

State:
S401        a>b,c>d,-(a>d),-(-a+d),a*-d,b,b*-d,-d>-c
            Action: 1         To State: 707     Prob: 25                   Times: 1
            Action: 3         To State: 407     Prob: 50                   Times: 2
            Action: 10        To State: 406     Prob: 25                   Times: 1

State:
S402        a>b,c>d,-(a>d),(a+c)>(b+d) 0

State:
S403        a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c,b*-b 0

State:
S404        a>b,c>d,-(a>d),-b>-a,-d>-c,-(-a+d),a*-d,-a>-d 0

State:
S405        a>b,c>d,-(a>d),-(-a+d),a*-d,b,-d>-c 0

State:
S406        a>b,c>d,-(a>d),-(-a+d),a*-d,b,b*-d,-d>-c,-c 0

State:
S407        a>b,c>d,-(a>d),-(-a+d),a*-d,b,b*-d,-d>-c,-c
            Action: 10        To State: 6       Prob: 100                  Times: 2

State:
S408        a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c,c 0

State:
S409        a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,b 0

State:
S410        a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,-c+d,b*-c 0

State:
S411        a>b,c>d,-(a>d),-(-d>-a),a*-d,-d>-c,-c+d,b*-c 0

State:
S412        a>b,c>d,-(a>d),a*-d,-c
            Action: 3         To State: 614     Prob: 12.5                 Times: 1
            Action: 3         To State: 413     Prob: 62.5                 Times: 5
            Action: 8         To State: 612     Prob: 12.5                 Times: 1
            Action: 33        To State: 613     Prob: 12.5                 Times: 1

State:      a>b,c>d,-(a>d),a*-d,-c,b
```

```
S413
            Action: 7          To State: 819      Prob: 20                         Times: 1
            Action: 10         To State: 6        Prob: 80                         Times: 4

State:
S414        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-a+b
            Action: 2          To State: 417      Prob: 20                         Times: 1
            Action: 2          To State: 416      Prob: 20                         Times: 1
            Action: 3          To State: 416      Prob: 20                         Times: 1
            Action: 10         To State: 415      Prob: 20                         Times: 1
            Action: 30         To State: 539      Prob: 20                         Times: 1

State:
S415        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-a+b,a 0

State:
S416        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-a+b,b 0

State:
S417        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-a+b,b
            Action: 2          To State: 419      Prob: 28.5714285714286           Times: 2
            Action: 2          To State: 418      Prob: 71.4285714285714           Times: 5

State:
S418        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-a+b,b,-c 0

State:
S419        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-a+b,b,-c
            Action: 10         To State: 6        Prob: 100                        Times: 2

State:
S420        a>b,c>d,-(a>d),(a*-d),a*-d
            Action: 3          To State: 421      Prob: 100                        Times: 1

State:
S421        a>b,c>d,-(a>d),(a*-d),a*-d,b
            Action: 10         To State: 6        Prob: 100                        Times: 1

State:
S422        a>b,c>d,-(a>d),(a*-d),-d>-c,a>-c 0

State:
S423        a>b,c>d,-(a>d),(a*-d),-d>-c,b*-c 0

State:
S424        a>b,c>d,-(a>d),(a*-d),-d>-c,b
            Action: 3          To State: 678      Prob: 16.6666666666667           Times: 2
            Action: 3          To State: 429      Prob: 25                         Times: 3
            Action: 3          To State: 425      Prob: 16.6666666666667           Times: 2
            Action: 8          To State: 427      Prob: 8.33333333333333           Times: 1
            Action: 8          To State: 426      Prob: 25                         Times: 3
            Action: 9          To State: 428      Prob: 8.33333333333333           Times: 1
```

```
State:
S425          a>b,c>d,-(a>d),(a*-d),-d>-c,b,b*-d 0

State:
S426          a>b,c>d,-(a>d),(a*-d),-d>-c,b,b*-c 0

State:
S427          a>b,c>d,-(a>d),(a*-d),-d>-c,b,b>d 0

State:
S428          a>b,c>d,-(a>d),(a*-d),-d>-c,b,(a>b)+(c>d) 0

State:
S429          a>b,c>d,-(a>d),(a*-d),-d>-c,b,-c
              Action: 10          To State: 6         Prob: 100                          Times: 3

State:
S430          a>b,c>d,-(a>d),a*-d,-c 0

State:
S431          a>b,c>d,-(a>d),a*-d,-(-a+b) 0

State:
S432          a>b,c>d,-(a>d),a*-d,-(a*-b)
              Action: 28          To State: 433       Prob: 100                          Times: 1

State:
S433          a>b,c>d,-(a>d),a*-d,-(a*-b),-a+b

State:
S434          a>b,c>d,-(a>d),-(-d>-a),-d>-c

State:
S435          a>b,c>d,-(a>d),-a+b,-c+d,-c 0

State:
S436          a>b,c>d,-(a>d),-a+b,-c+d,-(-d>-a)
              Action: 4           To State: 437       Prob: 25                           Times: 1
              Action: 30          To State: 643       Prob: 25                           Times: 1
              Action: 30          To State: 642       Prob: 25                           Times: 1
              Action: 30          To State: 438       Prob: 25                           Times: 1

State:
S437          a>b,c>d,-(a>d),-a+b,-c+d,-(-d>-a),d> 0

State:
S438          a>b,c>d,-(a>d),-a+b,-c+d,-(-d>-a),-(-a+d)
              Action: 27          To State: 440       Prob: 50                           Times: 1
              Action: 27          To State: 439       Prob: 50                           Times: 1

State:
S439          a>b,c>d,-(a>d),-a+b,-c+d,-(-d>-a),-(-a+d),a*-d 0
```

```
State:
S440      a>b,c>d,-(a>d),-a+b,-c+d,-(-d>-a),-(-a+d),a*-d
          Action: 9        To State: 441    Prob: 50                    Times: 1
          Action: 29       To State: 442    Prob: 50                    Times: 1

State:
S441      a>b,c>d,-(a>d),-a+b,-c+d,-(-d>-a),-(-a+d),a*-d,-d 0

State:
S442      a>b,c>d,-(a>d),-a+b,-c+d,-(-d>-a),-(-a+d),a*-d,-d>-c
          Action: 3        To State: 443    Prob: 100                   Times: 1

State:
S443      a>b,c>d,-(a>d),-a+b,-c+d,-(-d>-a),-(-a+d),a*-d,-d>-c,-c 0

State:
S444      a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,-c,-a+b
          Action: 2        To State: 446    Prob: 50                    Times: 1
          Action: 2        To State: 445    Prob: 50                    Times: 1

State:
S445      a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,-c,-a+b,b 0

State:
S446      a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,-c,-a+b,b
          Action: 10       To State: 6      Prob: 100                   Times: 1

State:    a>b,c>d,-(a>d),(a*c)>(b*d),-
S447      (a*c)+(b*d)
          Action: 28       To State: 448    Prob: 100                   Times: 1

State:
S448      a>b,c>d,-(a>d),(a*c)>(b*d),-(a*c)+(b*d),(-a+-c)+(b*d)
          Action: 30       To State: 449    Prob: 100                   Times: 1

State:
S449      a>b,c>d,-(a>d),(a*c)>(b*d),-(a*c)+(b*d),(-a+-c)+(b*d),-(-a+d)
          Action: 27       To State: 450    Prob: 100                   Times: 1

State:
S450      a>b,c>d,-(a>d),(a*c)>(b*d),-(a*c)+(b*d),(-a+-c)+(b*d),-(-a+d),a*-d
          Action: 2        To State: 452    Prob: 20                    Times: 1
          Action: 2        To State: 451    Prob: 20                    Times: 1
          Action: 4        To State: 453    Prob: 20                    Times: 1
          Action: 4        To State: 451    Prob: 40                    Times: 2

State:
S451      a>b,c>d,-(a>d),(a*c)>(b*d),-(a*c)+(b*d),(-a+-c)+(b*d),-(-a+d),a*-d,-c 0

State:
S452      a>b,c>d,-(a>d),(a*c)>(b*d),-(a*c)+(b*d),(-a+-c)+(b*d),-(-a+d),a*-d,b*-c 0
```

```
State:
S453        a>b,c>d,-(a>d),(a*c)>(b*d),-(a*c)+(b*d),(-a+-c)+(b*d),-(-a+d),a*-d,-c
            Action: 3          To State: 454     Prob: 100                       Times: 1

State:
S454        a>b,c>d,-(a>d),(a*c)>(b*d),-(a*c)+(b*d),(-a+-c)+(b*d),-(-a+d),a*-d,-c,b
            Action: 10         To State: 6       Prob: 100                       Times: 1

State:
S455        a>b,c>d,-(a>d),-(a*-b),-a+b,-(c*-d),-c+d,(a*-d)
            Action: 28         To State: 457     Prob: 33.3333333333333          Times: 1
            Action: 28         To State: 456     Prob: 66.6666666666667          Times: 2

State:
S456        a>b,c>d,-(a>d),-(a*-b),-a+b,-(c*-d),-c+d,(a*-d),-(-a+d) 0

State:
S457        a>b,c>d,-(a>d),-(a*-b),-a+b,-(c*-d),-c+d,(a*-d),a+-d 0

State:
S458        a>b,c>d,-(a>d),-(-d>-a),-(d+-a),-d*a
            Action: 3          To State: 460     Prob: 66.6666666666667          Times: 2
            Action: 3          To State: 459     Prob: 33.3333333333333          Times: 1

State:      a>b,c>d,-(a>d),-(-d>-a),-(d+-a),-d*a,b
S459        0

State:
S460        a>b,c>d,-(a>d),-(-d>-a),-(d+-a),-d*a,b
            Action: 4          To State: 461     Prob: 50                        Times: 1
            Action: 30         To State: 582     Prob: 50                        Times: 1

State:
S461        a>b,c>d,-(a>d),-(-d>-a),-(d+-a),-d*a,b,-c
            Action: 10         To State: 6       Prob: 100                       Times: 1

State:
S462        a>b,c>d,-(a>d),-(b>d)
            Action: 3          To State: 464     Prob: 50                        Times: 1
            Action: 3          To State: 463     Prob: 50                        Times: 1

State:
S463        a>b,c>d,-(a>d),-(b>d),-(b>c) 0

State:
S464        a>b,c>d,-(a>d),-(b>d),-(b>c)
            Action: 30         To State: 465     Prob: 100                       Times: 1

State:
S465        a>b,c>d,-(a>d),-(b>d),-(b>c),-(-b+c)
            Action: 27         To State: 6       Prob: 100                       Times: 1

State:      a>b,c>d,-(a>d),b>c 0
```

```
S466

State:
S467        a>b,c>d,-(a>d),b 0

State:
S468        a>b,c>d,-(a>d),--(a>-d) 0

State:
S469        a>b,c>d,-(a>d),-(-a+d),a*-d,-(c*-d)

State:
S470        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,a 0

State:
S471        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,-d>-c
            Action: 30        To State: 472      Prob: 100                   Times: 1

State:
S472        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,-d>-c,-c+d
            Action: 1         To State: 473      Prob: 50                    Times: 1
            Action: 4         To State: 474      Prob: 50                    Times: 1

State:
S473        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,-d>-c,-c+d,a*-c 0

State:
S474        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,-d>-c,-c+d,b*-c 0

State:
S475        a>b,c>d,-(a>d),-(-a+d),--a+-d 0

State:      a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,-b>-
S476        a
            Action: 30        To State: 478      Prob: 50                    Times: 1
            Action: 32        To State: 477      Prob: 50                    Times: 1

State:
S477        a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,-b>-a,-b*-d 0

State:
S478        a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,-b>-a,b+-a
            Action: 3         To State: 479      Prob: 20                    Times: 1
            Action: 27        To State: 479      Prob: 40                    Times: 2
            Action: 28        To State: 480      Prob: 40                    Times: 2

State:
S479        a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,-b>-a,b+-a,--b*-a 0

State:
S480        a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,-b>-a,b+-a,-(-b*a) 0

State:      a>b,c>d,-(a>d),(a*-d),a*-b 0
```

```
S481

State:
S482          a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,b
              Action: 30          To State: 483       Prob: 100                        Times: 3

State:
S483          a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,b,-c+d
              Action: 2           To State: 760       Prob: 40                         Times: 2
              Action: 2           To State: 717       Prob: 40                         Times: 2
              Action: 10          To State: 6         Prob: 20                         Times: 1

State:
S484          a>b,c>d,-(a>d),-(-a+d),a*-b 0

State:
S485          a>b,c>d,-(a>d),(a*c)>(b*d),-(-a+d),a*-d,b 0

State:
S486          a>b,c>d,-(a>d),(a*c)>(b*d),-(-a+d),a*-d,b
              Action: 4           To State: 487       Prob: 100                        Times: 3

State:
S487          a>b,c>d,-(a>d),(a*c)>(b*d),-(-a+d),a*-d,b,-c
              Action: 9           To State: 657       Prob: 25                         Times: 1
              Action: 10          To State: 6         Prob: 75                         Times: 3

State:
S488          a>b,c>d,-(a>d),-b>-a,-d>-c,(a*-d)
              Action: 28          To State: 489       Prob: 50                         Times: 1
              Action: 30          To State: 490       Prob: 50                         Times: 1

State:
S489          a>b,c>d,-(a>d),-b>-a,-d>-c,(a*-d),-a+d 0

State:
S490          a>b,c>d,-(a>d),-b>-a,-d>-c,(a*-d),-a+b
              Action: 30          To State: 491       Prob: 100                        Times: 1

State:
S491          a>b,c>d,-(a>d),-b>-a,-d>-c,(a*-d),-a+b,-c+d
              Action: 3           To State: 492       Prob: 100                        Times: 1

State:
S492          a>b,c>d,-(a>d),-b>-a,-d>-c,(a*-d),-a+b,-c+d,b
              Action: 4           To State: 493       Prob: 100                        Times: 1

State:
S493          a>b,c>d,-(a>d),-b>-a,-d>-c,(a*-d),-a+b,-c+d,b,-c
              Action: 10          To State: 6         Prob: 100                        Times: 1

State:
S494          a>b,c>d,-(a>d),a+-d 0
```

```
State:
S495          a>b,c>d,-(a>d),-(a*-b),-a+b,-(c*-d),-a*b 0

State:
S496          a>b,c>d,-(a>d),-(a*-b),-a+b,-(c*-d),-c+d,-a>-d 0

State:
S497          a>b,c>d,-(a>d),-(a*-b),-a+b,-(c*-d),-c+d,a*-d
              Action: 4          To State: 498    Prob: 100                 Times: 1

State:
S498          a>b,c>d,-(a>d),-(a*-b),-a+b,-(c*-d),-c+d,a*-d,-c
              Action: 29         To State: 500    Prob: 40                  Times: 2
              Action: 29         To State: 499    Prob: 60                  Times: 3

State:
S499          a>b,c>d,-(a>d),-(a*-b),-a+b,-(c*-d),-c+d,a*-d,-c,a>d 0

State:
S500          a>b,c>d,-(a>d),-(a*-b),-a+b,-(c*-d),-c+d,a*-d,-c,d>a 0

State:
S501          a>b,c>d,-(a>d),-(-a+d),a*-d 0

State:
S502          a>b,c>d,-(a>d),-(-a+d),a*-d,b*-d
              Action: 1          To State: 6      Prob: 50                  Times: 1
              Action: 7          To State: 503    Prob: 50                  Times: 1

State:
S503          a>b,c>d,-(a>d),-(-a+d),a*-d,b*-d,b*-c 0

State:
S504          a>b,c>d,-(a>d),-(-a+d),a*-d,-(b>d) 0

State:
S505          a>b,c>d,-(a>d),-(-a+d),a*-d,(a*c)>(b*d),-(a*c)+(b*d)
              Action: 3          To State: 684    Prob: 40                  Times: 2
              Action: 3          To State: 506    Prob: 40                  Times: 2
              Action: 27         To State: 542    Prob: 20                  Times: 1

State:
S506          a>b,c>d,-(a>d),-(-a+d),a*-d,(a*c)>(b*d),-(a*c)+(b*d),b
              Action: 4          To State: 507    Prob: 100                 Times: 2

State:
S507          a>b,c>d,-(a>d),-(-a+d),a*-d,(a*c)>(b*d),-(a*c)+(b*d),b,-c
              Action: 10         To State: 6      Prob: 100                 Times: 2

State:
S508          a>b,c>d,-(a>d),-(-a+d),a*-d,--a 0
```

```
State:
S509          a>b,c>d,-(a>d),(a*-d),-(-a+d),-(-a+d)
              Action: 27          To State: 510      Prob: 100                    Times: 1

State:
S510          a>b,c>d,-(a>d),(a*-d),-(-a+d),-(-a+d),a*-d
              Action: 30          To State: 511      Prob: 100                    Times: 1

State:
S511          a>b,c>d,-(a>d),(a*-d),-(-a+d),-(-a+d),a*-d,-a+b
              Action: 4           To State: 512      Prob: 100                    Times: 1

State:
S512          a>b,c>d,-(a>d),(a*-d),-(-a+d),-(-a+d),a*-d,-a+b,b
              Action: 2           To State: 513      Prob: 100                    Times: 1

State:
S513          a>b,c>d,-(a>d),(a*-d),-(-a+d),-(-a+d),a*-d,-a+b,b,b
              Action: 30          To State: 514      Prob: 100                    Times: 1

State:
S514          a>b,c>d,-(a>d),(a*-d),-(-a+d),-(-a+d),a*-d,-a+b,b,b,-c+d
              Action: 2           To State: 515      Prob: 100                    Times: 1

State:
S515          a>b,c>d,-(a>d),(a*-d),-(-a+d),-(-a+d),a*-d,-a+b,b,b,-c+d,-c
              Action: 10          To State: 6        Prob: 100                    Times: 1

State:
S516          a>b,c>d,-(a>d),-a+b,-c+d,a+-d 0

State:
S517          a>b,c>d,-(a>d),(a*-d),c 0

State:
S518          a>b,c>d,-(a>d),-d>-c,-(-a+d)
              Action: 27          To State: 519      Prob: 100                    Times: 2

State:
S519          a>b,c>d,-(a>d),-d>-c,-(-a+d),a*-d
              Action: 3           To State: 639      Prob: 50                     Times: 1
              Action: 3           To State: 520      Prob: 50                     Times: 1

State:
S520          a>b,c>d,-(a>d),-d>-c,-(-a+d),a*-d,b
              Action: 3           To State: 521      Prob: 100                    Times: 1

State:
S521          a>b,c>d,-(a>d),-d>-c,-(-a+d),a*-d,b,-c
              Action: 10          To State: 6        Prob: 100                    Times: 1

State:        a>b,c>d,-(a>d),-a+b,-(a*-b),-c+d,-(c*-
S522          d)
```

```
                Action: 31          To State: 525       Prob: 33.3333333333333          Times: 1
                Action: 31          To State: 524       Prob: 33.3333333333333          Times: 1
                Action: 31          To State: 523       Prob: 33.3333333333333          Times: 1

State:
S523            a>b,c>d,-(a>d),-a+b,-(a*-b),-c+d,-(c*-d),--(a*-d) 0

State:
S524            a>b,c>d,-(a>d),-a+b,-(a*-b),-c+d,-(c*-d),--(a*d) 0

State:
S525            a>b,c>d,-(a>d),-a+b,-(a*-b),-c+d,-(c*-d),(a*-d)
                Action: 30          To State: 526       Prob: 100                       Times: 1

State:
S526            a>b,c>d,-(a>d),-a+b,-(a*-b),-c+d,-(c*-d),(a*-d),b*-c 0

State:
S527            a>b,c>d,-(a>d),(a>b)+-(a>d) 0

State:
S528            a>b,c>d,-(a>d),b*-d 0

State:
S529            a>b,c>d,-(a>d),b*-c 0

State:
S530            a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,b
                Action: 3           To State: 531       Prob: 66.6666666666667          Times: 2
                Action: 4           To State: 531       Prob: 33.3333333333333          Times: 1

State:
S531            a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,b,-c
                Action: 10          To State: 801       Prob: 33.3333333333333          Times: 1
                Action: 10          To State: 6         Prob: 66.6666666666667          Times: 2

State:
S532            a>b,c>d,-(a>d),a*-d,-a+b
                Action: 2           To State: 533       Prob: 100                       Times: 1

State:
S533            a>b,c>d,-(a>d),a*-d,-a+b,b
                Action: 4           To State: 534       Prob: 100                       Times: 1

State:
S534            a>b,c>d,-(a>d),a*-d,-a+b,b,-c
                Action: 10          To State: 6         Prob: 100                       Times: 1

State:
S535            a>b,c>d,-(a>d),-a>-d

State:
S536            a>b,c>d,-(a>d),-(-a*d) 0
```

```
State:
S537          a>b,c>d,-(a>d),b

State:        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-a*b
S538          0

State:
S539          a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-a+b,-a>-d 0

State:
S540          a>b,c>d,-(a>d),-(-a+-d) 0

State:
S541          a>b,c>d,-(a>d),-(-a+d),--a+d 0

State:
S542          a>b,c>d,-(a>d),-(-a+d),a*-d,(a*c)>(b*d),-(a*c)+(b*d),(-a+-c)+(b*d)

State:
S543          a>b,c>d,-(a>d),-(a>d)
              Action: 30          To State: 544       Prob: 100                         Times: 2

State:
S544          a>b,c>d,-(a>d),-(a>d),-(-a+d)
              Action: 27          To State: 545       Prob: 100                         Times: 2

State:
S545          a>b,c>d,-(a>d),-(a>d),-(-a+d),a*-d
              Action: 3           To State: 547       Prob: 50                          Times: 2
              Action: 3           To State: 546       Prob: 50                          Times: 2

State:
S546          a>b,c>d,-(a>d),-(a>d),-(-a+d),a*-d,b 0

State:
S547          a>b,c>d,-(a>d),-(a>d),-(-a+d),a*-d,b
              Action: 4           To State: 549       Prob: 40                          Times: 2
              Action: 4           To State: 548       Prob: 60                          Times: 3

State:
S548          a>b,c>d,-(a>d),-(a>d),-(-a+d),a*-d,b,-c 0

State:
S549          a>b,c>d,-(a>d),-(a>d),-(-a+d),a*-d,b,-c
              Action: 10          To State: 6         Prob: 100                         Times: 2

State:
S550          a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,-c,-c

State:
S551          a>b,c>d,-(a>d),-b>-a,-a+b
              Action: 27          To State: 781       Prob: 50                          Times: 1
```

```
                  Action: 30          To State: 552      Prob: 50                     Times: 1

State:
S552              a>b,c>d,-(a>d),-b>-a,-a+b,-c+d
                  Action: 7           To State: 553      Prob: 100                    Times: 1

State:
S553              a>b,c>d,-(a>d),-b>-a,-a+b,-c+d,(-b+c)>(-a+d)
                  Action: 5           To State: 555      Prob: 33.3333333333333       Times: 1
                  Action: 5           To State: 554      Prob: 33.3333333333333       Times: 1
                  Action: 30          To State: 556      Prob: 33.3333333333333       Times: 1

State:
S554              a>b,c>d,-(a>d),-b>-a,-a+b,-c+d,(-b+c)>(-a+d),-b+c 0

State:
S555              a>b,c>d,-(a>d),-b>-a,-a+b,-c+d,(-b+c)>(-a+d),-(-b+c) 0

State:
S556              a>b,c>d,-(a>d),-b>-a,-a+b,-c+d,(-b+c)>(-a+d),-(-a+d)
                  Action: 4           To State: 557      Prob: 100                    Times: 1

State:
S557              a>b,c>d,-(a>d),-b>-a,-a+b,-c+d,(-b+c)>(-a+d),-(-a+d),-(-b+c)
                  Action: 28          To State: 6        Prob: 100                    Times: 1

State:
S558              a>b,c>d,-(a>d),-a+b,-c+-d 0

State:
S559              a>b,c>d,-(a>d),-a+b,c-+d 0

State:
S560              a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,-b>-a
                  Action: 27          To State: 562      Prob: 66.6666666666667       Times: 2
                  Action: 30          To State: 561      Prob: 33.3333333333333       Times: 1

State:
S561              a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,-b>-a,-a>-d 0

State:
S562              a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,-b>-a,-a+d 0

State:
S563              a>b,c>d,-(a>d),-a+b,-c>d 0

State:
S564              a>b,c>d,-(a>d),--(-a+d) 0

State:
S565              a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d,b,-c,b*-d 0

State:             a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-d 0
```

```
S566

State:
S567          a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-c,-a+b,a 0

State:
S568          a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-c,-a+b,b,b*-c 0

State:
S569          a>b,c>d,-(a>d),c>-d 0

State:
S570          a>b,c>d,-(a>d),c>-a 0

State:
S571          a>b,c>d,-(a>d),b>-d 0

State:
S572          a>b,c>d,-(a>d),a*-d,b*-d 0

State:
S573          a>b,c>d,-(a>d),--(a*d) 0

State:
S574          a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c+d,-c>-d 0

State:
S575          a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c+d,-d>-c
              Action: 3           To State: 821      Prob: 50                          Times: 1
              Action: 30          To State: 576      Prob: 50                          Times: 1

State:
S576          a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c+d,-d>-c,-c+d
              Action: 3           To State: 577      Prob: 100                         Times: 1

State:
S577          a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c+d,-d>-c,-c+d,-c
              Action: 10          To State: 6        Prob: 100                         Times: 1

State:
S578          a>b,c>d,-(a>d),(a*-d),b,b*-d 0

State:
S579          a>b,c>d,-(a>d),(a*-d),b,-(c*-d)
              Action: 30          To State: 580      Prob: 100                         Times: 1

State:
S580          a>b,c>d,-(a>d),(a*-d),b,-(c*-d),-c+d

State:        a>b,c>d,-(a>d),-(-d>-a),-(d+-a),-d+--a
S581          0

State:        a>b,c>d,-(a>d),-(-d>-a),-(d+-a),-d*a,b,-c+d
```

```
S582

State:
S583          a>b,c>d,-(a>d),-(a+-d) 0

State:
S584          a>b,c>d,-(a>d),(a*-d),-c,-b>-a
              Action: 4          To State: 587      Prob: 11.1111111111111         Times: 1
              Action: 4          To State: 586      Prob: 11.1111111111111         Times: 1
              Action: 4          To State: 585      Prob: 77.7777777777778         Times: 7

State:
S585          a>b,c>d,-(a>d),(a*-d),-c,-b>-a,--b 0

State:
S586          a>b,c>d,-(a>d),(a*-d),-c,-b>-a,b 0

State:
S587          a>b,c>d,-(a>d),(a*-d),-c,-b>-a,b
              Action: 10         To State: 6        Prob: 100                      Times: 1

State:
S588          a>b,c>d,-(a>d),a+c 0

State:
S589          a>b,c>d,-(a>d),(a+c)>(b+d),(a+c)>b 0

State:
S590          a>b,c>d,-(a>d),(a+c)>(b+d),-a+-d 0

State:
S591          a>b,c>d,-(a>d),(a+c)>(b+d),a+-d 0

State:
S592          a>b,c>d,-(a>d),(a+c)>(b+d),-c+d
              Action: 29         To State: 593      Prob: 50                       Times: 1
              Action: 30         To State: 594      Prob: 50                       Times: 1

State:
S593          a>b,c>d,-(a>d),(a+c)>(b+d),-c+d,-a+-d 0

State:
S594          a>b,c>d,-(a>d),(a+c)>(b+d),-c+d,-(-a+d)
              Action: 30         To State: 595      Prob: 100                      Times: 1

State:
S595          a>b,c>d,-(a>d),(a+c)>(b+d),-c+d,-(-a+d),-a+b
              Action: 2          To State: 596      Prob: 50                       Times: 1
              Action: 30         To State: 597      Prob: 50                       Times: 1

State:
S596          a>b,c>d,-(a>d),(a+c)>(b+d),-c+d,-(-a+d),-a+b,b 0
```

```
State:
S597        a>b,c>d,-(a>d),(a+c)>(b+d),-c+d,-(-a+d),-a+b,-(a+c)+(b+d)

State:
S598        a>b,c>d,-(a>d),-(-a+d),-a+b,b
            Action: 30          To State: 599       Prob: 100                   Times: 1

State:
S599        a>b,c>d,-(a>d),-(-a+d),-a+b,b,-c+d
            Action: 2           To State: 600       Prob: 100                   Times: 1

State:
S600        a>b,c>d,-(a>d),-(-a+d),-a+b,b,-c+d,-c
            Action: 10          To State: 6         Prob: 100                   Times: 1

State:
S601        a>b,c>d,-(a>d),-a+b,c>d 0

State:      a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,a*-b
S602        0

State:      a>b,c>d,-(a>d),-b
S603        0

State:
S604        a>b,c>d,-(a>d),-(-a+d),a*-d,-c,-c*a 0

State:
S605        a>b,c>d,-(a>d),-(-a+d),a*-d,-a*b 0

State:
S606        a>b,c>d,-(a>d),-a+b,-c+d,a*-d,-c 0

State:
S607        a>b,c>d,-(a>d),-a+b,-c+d,a*-d,d*-d 0

State:
S608        a>b,c>d,-(a>d),-a+b,-c+d,a*-d,-a*a 0

State:
S609        a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,b*-d 0

State:
S610        a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c 0

State:
S611        a>b,c>d,-(a>d),-(-a+d),a*-d,a>b 0

State:
S612        a>b,c>d,-(a>d),a*-d,-c,b*-c 0

State:
S613        a>b,c>d,-(a>d),a*-d,-c,a>(b*-c) 0
```

```
State:
S614        a>b,c>d,-(a>d),a*-d,-c,b 0

State:
S615        a>b,c>d,-(a>d),b+c 0

State:
S616        a>b,c>d,-(a>d),-(-a+d),-(-a+d)
            Action: 27        To State: 617     Prob: 100                  Times: 1

State:
S617        a>b,c>d,-(a>d),-(-a+d),-(-a+d),a*-d
            Action: 3         To State: 619     Prob: 50                   Times: 1
            Action: 3         To State: 618     Prob: 50                   Times: 1

State:
S618        a>b,c>d,-(a>d),-(-a+d),-(-a+d),a*-d,b 0

State:
S619        a>b,c>d,-(a>d),-(-a+d),-(-a+d),a*-d,b
            Action: 29        To State: 620     Prob: 100                  Times: 1

State:
S620        a>b,c>d,-(a>d),-(-a+d),-(-a+d),a*-d,b,-d>-c
            Action: 3         To State: 621     Prob: 100                  Times: 1

State:
S621        a>b,c>d,-(a>d),-(-a+d),-(-a+d),a*-d,b,-d>-c,-c
            Action: 10        To State: 6       Prob: 100                  Times: 1

State:
S622        a>b,c>d,-(a>d),(a*-d),b*-c 0

State:      a>b,c>d,-(a>d),(a*-d),(a*c)>(b*d),a>d
S623        0

State:
S624        a>b,c>d,-(a>d),(a*-d),(a*c)>(b*d),-c>-d 0

State:      a>b,c>d,-(a>d),(a*-d),(a*c)>(b*d),-b>-
S625        a
            Action: 29        To State: 626     Prob: 100                  Times: 1

State:
S626        a>b,c>d,-(a>d),(a*-d),(a*c)>(b*d),-b>-a,-d>-c

State:
S627        a>b,c>d,-(a>d),-a+b,-d>-c
            Action: 31        To State: 628     Prob: 100                  Times: 1

State:
S628        a>b,c>d,-(a>d),-a+b,-d>-c,a*-d
            Action: 3         To State: 629     Prob: 100                  Times: 1
```

```
State:
S629       a>b,c>d,-(a>d),-a+b,-d>-c,a*-d,-c
           Action: 3          To State: 630     Prob: 100                      Times: 1

State:
S630       a>b,c>d,-(a>d),-a+b,-d>-c,a*-d,-c,b
           Action: 10         To State: 6       Prob: 100                      Times: 1

State:
S631       a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c,b+-c 0

State:
S632       a>b,c>d,-(a>d),(a*c)>(b*d),-(-a+d),a*-d,-d 0

State:
S633       a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,b*-c 0

State:
S634       a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,b,-c+d
           Action: 2          To State: 635     Prob: 66.6666666666667         Times: 2
           Action: 5          To State: 708     Prob: 33.3333333333333         Times: 1

State:
S635       a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,b,-c+d,-c
           Action: 10         To State: 6       Prob: 100                      Times: 2

State:
S636       a>b,c>d,-(a>d),-(-a+d),(a*-d)
           Action: 3          To State: 637     Prob: 100                      Times: 1

State:
S637       a>b,c>d,-(a>d),-(-a+d),(a*-d),b
           Action: 4          To State: 638     Prob: 100                      Times: 1

State:
S638       a>b,c>d,-(a>d),-(-a+d),(a*-d),b,-c
           Action: 10         To State: 6       Prob: 100                      Times: 1

State:
S639       a>b,c>d,-(a>d),-d>-c,-(-a+d),a*-d,-c
           Action: 3          To State: 640     Prob: 100                      Times: 1

State:
S640       a>b,c>d,-(a>d),-d>-c,-(-a+d),a*-d,-c,b
           Action: 10         To State: 6       Prob: 100                      Times: 1

State:
S641       a>b,c>d,-(a>d),-a+b,-c+d,-(a*-d) 0

State:
S642       a>b,c>d,-(a>d),-a+b,-c+d,-(-d>-a),-(d>-a) 0
```

```
State:
S643     a>b,c>d,-(a>d),-a+b,-c+d,-(-d>-a),-(d+-a)
         Action: 31        To State: 644    Prob: 100                       Times: 3

State:
S644     a>b,c>d,-(a>d),-a+b,-c+d,-(-d>-a),-(d+-a),-(-d*a) 0

State:
S645     a>b,c>d,-(a>d),-d>-c,-a>-d 0

State:
S646     a>b,c>d,-(a>d),-d>-c,a*-d,(-d*a)>(-c*b) 0

State:   a>b,c>d,-(a>d),-d>-c,a*-d,(a*-d)>(b*-
S647     c)
         Action: 3         To State: 6      Prob: 33.3333333333333          Times: 1
         Action: 3         To State: 648    Prob: 66.6666666666667          Times: 2

State:
S648     a>b,c>d,-(a>d),-d>-c,a*-d,(a*-d)>(b*-c),b*-c 0

State:
S649     a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,b 0

State:
S650     a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,b
         Action: 4         To State: 652    Prob: 50                        Times: 1
         Action: 4         To State: 651    Prob: 50                        Times: 1

State:
S651     a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,b,-c 0

State:
S652     a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,b,-c
         Action: 10        To State: 6      Prob: 100                       Times: 1

State:
S653     a>b,c>d,-(a>d),c>b 0

State:
S654     a>b,c>d,-(a>d),-a+b,-c+d,b+-c 0

State:
S655     a>b,c>d,-(a>d),-a+b,-c+d,b+-c
         Action: 10        To State: 6      Prob: 100                       Times: 1

State:
S656     a>b,c>d,-(a>d),(a*c)>(b*d),(a*d)>(b*d) 0

State:
S657     a>b,c>d,-(a>d),(a*c)>(b*d),-(-a+d),a*-d,b,-c,b*-c 0

State:   a>b,c>d,-(a>d),-(a+-b) 0
```

```
S658

State:
S659        a>b,c>d,-(a>d),a*-d,b,-d>-c
            Action: 3          To State: 660     Prob: 100               Times: 1

State:
S660        a>b,c>d,-(a>d),a*-d,b,-d>-c,-c
            Action: 10         To State: 6       Prob: 100               Times: 1

State:
S661        a>b,c>d,-(a>d),(a+c)>(b+d),-(a+c)+(b+d)
            Action: 30         To State: 662     Prob: 100               Times: 1

State:
S662        a>b,c>d,-(a>d),(a+c)>(b+d),-(a+c)+(b+d),-(-a+d)
            Action: 27         To State: 664     Prob: 50                Times: 1
            Action: 27         To State: 663     Prob: 50                Times: 1

State:
S663        a>b,c>d,-(a>d),(a+c)>(b+d),-(a+c)+(b+d),-(-a+d),a+-d 0

State:
S664        a>b,c>d,-(a>d),(a+c)>(b+d),-(a+c)+(b+d),-(-a+d),a*-d
            Action: 3          To State: 666     Prob: 50                Times: 1
            Action: 3          To State: 665     Prob: 50                Times: 1

State:
S665        a>b,c>d,-(a>d),(a+c)>(b+d),-(a+c)+(b+d),-(-a+d),a*-d,b 0

State:
S666        a>b,c>d,-(a>d),(a+c)>(b+d),-(a+c)+(b+d),-(-a+d),a*-d,b
            Action: 4          To State: 668     Prob: 50                Times: 1
            Action: 4          To State: 667     Prob: 50                Times: 1

State:
S667        a>b,c>d,-(a>d),(a+c)>(b+d),-(a+c)+(b+d),-(-a+d),a*-d,b,c 0

State:
S668        a>b,c>d,-(a>d),(a+c)>(b+d),-(a+c)+(b+d),-(-a+d),a*-d,b,-c
            Action: 10         To State: 6       Prob: 100               Times: 1

State:
S669        a>b,c>d,-(a>d),-c+d,-(a*-b)

State:
S670        a>b,c>d,-(a>d),-a+b,a*-d,-c
            Action: 10         To State: 6       Prob: 100               Times: 1

State:
S671        a>b,c>d,-(a>d),-b>-a,b*-a 0

State:     a>b,c>d,-(a>d),-c>-d 0
```

```
S672

State:
S673        a>b,c>d,-(a>d),-d>-c,(a*-d)
            Action: 3          To State: 727     Prob: 60                    Times: 3
            Action: 3          To State: 674     Prob: 40                    Times: 2

State:
S674        a>b,c>d,-(a>d),-d>-c,(a*-d),b
            Action: 3          To State: 675     Prob: 100                   Times: 2

State:
S675        a>b,c>d,-(a>d),-d>-c,(a*-d),b,-c
            Action: 10         To State: 6       Prob: 100                   Times: 2

State:
S676        a>b,c>d,-(a>d),(a*c)>(b*d),-(-a+d),a*-d,-(a*c)+(b*d)

State:
S677        a>b,c>d,-(a>d),(a*-d),-d>-c,b*-d 0

State:
S678        a>b,c>d,-(a>d),(a*-d),-d>-c,b,-c 0

State:
S679        a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,b>-c 0

State:
S680        a>b,c>d,-(a>d),a*-d,-d>-c,b>d 0

State:
S681        a>b,c>d,-(a>d),a*-d,-d>-c,b
            Action: 3          To State: 682     Prob: 100                   Times: 1

State:
S682        a>b,c>d,-(a>d),a*-d,-d>-c,b,-c
            Action: 9          To State: 6       Prob: 50                    Times: 1
            Action: 10         To State: 683     Prob: 50                    Times: 1

State:
S683        a>b,c>d,-(a>d),a*-d,-d>-c,b,-c,b*-c 0

State:
S684        a>b,c>d,-(a>d),-(-a+d),a*-d,(a*c)>(b*d),-(a*c)+(b*d),b 0

State:
S685        a>b,c>d,-(a>d),-(-a+d),a*-d,-(a+b) 0

State:
S686        a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,(a+c)>(b+d) 0

State:
S687        a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,a*-d
```

```
                Action: 9          To State: 689    Prob: 50                    Times: 1
                Action: 30         To State: 688    Prob: 50                    Times: 1

State:
S688            a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,a*-d,d>a 0

State:
S689            a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,-c+d,a*-d,-d+a 0

State:
S690            a>b,c>d,-(a>d),(a*-d),-c+d
                Action: 2          To State: 691    Prob: 100                   Times: 1

State:
S691            a>b,c>d,-(a>d),(a*-d),-c+d,-c
                Action: 30         To State: 692    Prob: 100                   Times: 1

State:
S692            a>b,c>d,-(a>d),(a*-d),-c+d,-c,-a+b
                Action: 2          To State: 693    Prob: 100                   Times: 1

State:
S693            a>b,c>d,-(a>d),(a*-d),-c+d,-c,-a+b,b
                Action: 10         To State: 6      Prob: 100                   Times: 1

State:
S694            a>b,c>d,-(a>d),a+d 0

State:
S695            a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-b>-a
                Action: 27         To State: 696    Prob: 100                   Times: 1

State:
S696            a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-b>-a,a*-d
                Action: 29         To State: 699    Prob: 33.3333333333333      Times: 1
                Action: 31         To State: 698    Prob: 33.3333333333333      Times: 1
                Action: 31         To State: 697    Prob: 33.3333333333333      Times: 1

State:
S697            a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-b>-a,a*-d,a>d 0

State:
S698            a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-b>-a,a*-d,-(-d>a) 0

State:
S699            a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-b>-a,a*-d,-(-d>-a)
                Action: 1          To State: 700    Prob: 50                    Times: 1
                Action: 29         To State: 701    Prob: 50                    Times: 1

State:
S700            a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-b>-a,a*-d,-(-d>-a),-(-d>b) 0

State:          a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-b>-a,a*-d,-(-d>-a),-d>-c
```

```
S701
               Action: 3          To State: 703    Prob: 50                 Times: 1
               Action: 4          To State: 702    Prob: 50                 Times: 1

State:
S702           a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-b>-a,a*-d,-(-d>-a),-d>-c,-(b>d) 0

State:
S703           a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-b>-a,a*-d,-(-d>-a),-d>-c,-c
               Action: 3          To State: 704    Prob: 100                Times: 1

State:
S704           a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-b>-a,a*-d,-(-d>-a),-d>-c,-c,b
               Action: 10         To State: 6      Prob: 100                Times: 1

State:
S705           a>b,c>d,-(a>d),-b>c 0

State:
S706           a>b,c>d,-(a>d),(a*-d),-c 0

State:
S707           a>b,c>d,-(a>d),-(-a+d),a*-d,b,b*-d,-d>-c,b*-c 0

State:
S708           a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,b,-c+d,-d 0

State:
S709           a>b,c>d,-(a>d),-(-a+d),a*-d,-(-d>a) 0

State:
S710           a>b,c>d,-(a>d),-(-a+d),-d>-c
               Action: 27         To State: 711    Prob: 100                Times: 1

State:
S711           a>b,c>d,-(a>d),-(-a+d),-d>-c,a*-d
               Action: 3          To State: 714    Prob: 20                 Times: 1
               Action: 3          To State: 713    Prob: 60                 Times: 3
               Action: 10         To State: 712    Prob: 20                 Times: 1

State:
S712           a>b,c>d,-(a>d),-(-a+d),-d>-c,a*-d,b*-c 0

State:
S713           a>b,c>d,-(a>d),-(-a+d),-d>-c,a*-d,b 0

State:
S714           a>b,c>d,-(a>d),-(-a+d),-d>-c,a*-d,b
               Action: 3          To State: 715    Prob: 100                Times: 1

State:
S715           a>b,c>d,-(a>d),-(-a+d),-d>-c,a*-d,b,-c
               Action: 10         To State: 6      Prob: 100                Times: 1
```

```
State:
S716          a>b,c>d,-(a>d),-(a+b) 0

State:
S717          a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,b,-c+d,-c
              Action: 10          To State: 6       Prob: 100                         Times: 2

State:
S718          a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,c 0

State:
S719          a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,-d 0

State:
S720          a>b,c>d,-(a>d),(a*-d),-d 0

State:
S721          a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,c+-d
              Action: 2           To State: 722     Prob: 33.3333333333333            Times: 1
              Action: 29          To State: 724     Prob: 33.3333333333333            Times: 1
              Action: 30          To State: 723     Prob: 33.3333333333333            Times: 1

State:
S722          a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,c+-d,-c 0

State:
S723          a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,c+-d,d>c 0

State:
S724          a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,c+-d,-c>-d
              Action: 29          To State: 725     Prob: 100                         Times: 1

State:
S725          a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,c+-d,-c>-d,-d>-c
              Action: 3           To State: 726     Prob: 100                         Times: 1

State:
S726          a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,b,c+-d,-c>-d,-d>-c,-c
              Action: 10          To State: 6       Prob: 100                         Times: 1

State:
S727          a>b,c>d,-(a>d),-d>-c,(a*-d),b*-d 0

State:
S728          a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),-(-a+-c)
              Action: 27          To State: 731     Prob: 33.3333333333333            Times: 1
              Action: 27          To State: 730     Prob: 33.3333333333333            Times: 1
              Action: 27          To State: 729     Prob: 33.3333333333333            Times: 1

State:
S729          a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),-(-a+-c),a*c 0
```

```
State:
S730        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),-(-a+-c),a*d 0

State:
S731        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),-(-a+-c),a*-d
            Action: 29      To State: 732     Prob: 100                  Times: 1

State:
S732        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),-(-a+-c),a*-d,-d>-c
            Action: 3       To State: 733     Prob: 100                  Times: 1

State:
S733        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),-(-a+-c),a*-d,-d>-c,-c
            Action: 3       To State: 734     Prob: 33.3333333333333     Times: 1
            Action: 10      To State: 6       Prob: 33.3333333333333     Times: 1
            Action: 10      To State: 735     Prob: 33.3333333333333     Times: 1

State:
S734        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),-(-a+-c),a*-d,-d>-c,-c,b 0

State:
S735        a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),-(-a+-c),a*-d,-d>-c,-c,b*-c 0

State:
S736        a>b,c>d,-(a>d),-a+b,(a*-d),b 0

State:
S737        a>b,c>d,-(a>d),-a+b,(a*-d),b
            Action: 4       To State: 738     Prob: 100                  Times: 1

State:
S738        a>b,c>d,-(a>d),-a+b,(a*-d),b,-c
            Action: 10      To State: 6       Prob: 100                  Times: 1

State:
S739        a>b,c>d,-(a>d),-d>-c,-b>-a
            Action: 30      To State: 740     Prob: 100                  Times: 1

State:
S740        a>b,c>d,-(a>d),-d>-c,-b>-a,-(-a+d)
            Action: 27      To State: 741     Prob: 100                  Times: 1

State:      a>b,c>d,-(a>d),-d>-c,-b>-a,-(-a+d),a*-
S741        d
            Action: 3       To State: 742     Prob: 100                  Times: 1

State:
S742        a>b,c>d,-(a>d),-d>-c,-b>-a,-(-a+d),a*-d,b
            Action: 4       To State: 743     Prob: 100                  Times: 1

State:
S743        a>b,c>d,-(a>d),-d>-c,-b>-a,-(-a+d),a*-d,b,-c
            Action: 10      To State: 6       Prob: 50                   Times: 1
```

```
                     Action: 10              To State: 744        Prob: 50                                Times: 1

State:
S744                 a>b,c>d,-(a>d),-d>-c,-b>-a,-(-a+d),a*-d,b,-c,b*-c 0

State:
S745                 a>b,c>d,-(a>d),-a+b,-(-a+b) 0

State:
S746                 a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,-d+b 0

State:
S747                 a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,--d+b 0

State:
S748                 a>b,c>d,-(a>d),-c+d,b*-c 0

State:
S749                 a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d,$- 0

State:                a>b,c>d,-(a>d),-c+d,-(-a+d),a*-d,b*-d
S750                 0

State:
S751                 a>b,c>d,-(a>d),-(a*-b),-(c*-d),-a+b,-c+d,-(-a+d),a*-d,b 0

State:
S752                 a>b,c>d,-(a>d),a*-d,b,-c+d,-c 0

State:                a>b,c>d,-(a>d),-a
S753                 0

State:
S754                 a>b,c>d,-(a>d),-d>-c,(a*-d)>(b*-c) 0

State:
S755                 a>b,c>d,-(a>d),-d>-c,(a*-d)>(b*-c),-(-a+d)
                     Action: 27              To State: 756        Prob: 100                               Times: 1

State:
S756                 a>b,c>d,-(a>d),-d>-c,(a*-d)>(b*-c),-(-a+d),a*-d
                     Action: 3               To State: 6          Prob: 100                               Times: 1

State:
S757                 a>b,c>d,-(a>d),(a*c)>(b*d),-c+d

State:
S758                 a>b,c>d,-(a>d),-a+b,-c+d,-(-a+d),a*-d,c*-d 0

State:
S759                 a>b,c>d,-(a>d),a*-d,b,-c,b*-c 0

State:
S760                 a>b,c>d,-(a>d),-(-a+d),a*-d,-a+b,b,-c+d,-c 0
```

```
State:
S761          a>b,c>d,-(a>d),-b>-(a*-c) 0

State:
S762          a>b,c>d,-(a>d),-d>-c,-a*b 0

State:
S763          a>b,c>d,-(a>d),-(a*-b),-a+b,-c+d
              Action: 30          To State: 764      Prob: 100                    Times: 1

State:
S764          a>b,c>d,-(a>d),-(a*-b),-a+b,-c+d,-(-a+d)
              Action: 31          To State: 765      Prob: 100                    Times: 1

State:
S765          a>b,c>d,-(a>d),-(a*-b),-a+b,-c+d,-(-a+d),a*-d
              Action: 1           To State: 766      Prob: 50                     Times: 1
              Action: 2           To State: 767      Prob: 50                     Times: 1

State:
S766          a>b,c>d,-(a>d),-(a*-b),-a+b,-c+d,-(-a+d),a*-d,a 0

State:
S767          a>b,c>d,-(a>d),-(a*-b),-a+b,-c+d,-(-a+d),a*-d,-c
              Action: 2           To State: 768      Prob: 100                    Times: 1

State:
S768          a>b,c>d,-(a>d),-(a*-b),-a+b,-c+d,-(-a+d),a*-d,-c,b
              Action: 10          To State: 6        Prob: 100                    Times: 1

State:
S769          a>b,c>d,-(a>d),$ 0

State:
S770          a>b,c>d,-(a>d),(b>c)=-(b*-c) 0

State:
S771          a>b,c>d,-(a>d),(a*c)>(b*d),-(a*-d) 0

State:
S772          a>b,c>d,-(a>d),(a*c)>(b*d),a*-d
              Action: 4           To State: 773      Prob: 100                    Times: 1

State:
S773          a>b,c>d,-(a>d),(a*c)>(b*d),a*-d,-c
              Action: 3           To State: 775      Prob: 50                     Times: 1
              Action: 3           To State: 774      Prob: 50                     Times: 1

State:
S774          a>b,c>d,-(a>d),(a*c)>(b*d),a*-d,-c,b 0

State:        a>b,c>d,-(a>d),(a*c)>(b*d),a*-d,-c,b
```

```
S775

State:
S776        a>b,c>d,-(a>d),-a+b,-(-a+d),a*-d,b,-c,b*-c 0

State:
S777        a>b,c>d,-(a>d),-a+b,-c+d,a*-d,b,- 0

State:
S778        a>b,c>d,-(a>d),-a+b,-c+d,a*-d,b,-c 0

State:
S779        a>b,c>d,-(a>d),(a*-d),b,-c+d
            Action: 2           To State: 780     Prob: 100                      Times: 1

State:
S780        a>b,c>d,-(a>d),(a*-d),b,-c+d,-c
            Action: 10          To State: 6       Prob: 100                      Times: 1

State:
S781        a>b,c>d,-(a>d),-b>-a,-a+b,-(-b*a)

State:
S782        a>b,c>d,-(a>d),-(-a+d),a*-d,a*-b 0

State:
S783        a>b,c>d,-(a>d),a 0

State:
S784        a>b,c>d,-(a>d),a>-d 0

State:
S785        a>b,c>d,-(a>d),-d>a 0

State:
S786        a>b,c>d,-(a>d),-b>-a,-(-a+d)
            Action: 27          To State: 788     Prob: 50                       Times: 1
            Action: 27          To State: 787     Prob: 50                       Times: 1

State:
S787        a>b,c>d,-(a>d),-b>-a,-(-a+d),--a+-d 0

State:
S788        a>b,c>d,-(a>d),-b>-a,-(-a+d),a*-d
            Action: 29          To State: 790     Prob: 50                       Times: 1
            Action: 33          To State: 789     Prob: 50                       Times: 1

State:
S789        a>b,c>d,-(a>d),-b>-a,-(-a+d),a*-d,a 0

State:      a>b,c>d,-(a>d),-b>-a,-(-a+d),a*-d,-d>-
S790        c
            Action: 3           To State: 792     Prob: 50                       Times: 1
```

```
                 Action: 3        To State: 791     Prob: 50                     Times: 1

State:
S791             a>b,c>d,-(a>d),-b>-a,-(-a+d),a*-d,-d>-c,-c 0

State:
S792             a>b,c>d,-(a>d),-b>-a,-(-a+d),a*-d,-d>-c,b
                 Action: 3        To State: 793     Prob: 100                    Times: 1

State:
S793             a>b,c>d,-(a>d),-b>-a,-(-a+d),a*-d,-d>-c,b,-c
                 Action: 10       To State: 6       Prob: 100                    Times: 1

State:
S794             a>b,c>d,-(a>d),-(-a+d),-(c*-d)
                 Action: 29       To State: 795     Prob: 100                    Times: 1

State:
S795             a>b,c>d,-(a>d),-(-a+d),-(c*-d),-b>-a
                 Action: 2        To State: 798     Prob: 25                     Times: 1
                 Action: 2        To State: 797     Prob: 25                     Times: 1
                 Action: 4        To State: 796     Prob: 50                     Times: 2

State:           a>b,c>d,-(a>d),-(-a+d),-(c*-d),-b>-a,b
S796             0

State:
S797             a>b,c>d,-(a>d),-(-a+d),-(c*-d),-b>-a,-c 0

State:
S798             a>b,c>d,-(a>d),-(-a+d),-(c*-d),-b>-a,-c
                 Action: 3        To State: 799     Prob: 100                    Times: 1

State:
S799             a>b,c>d,-(a>d),-(-a+d),-(c*-d),-b>-a,-c,b
                 Action: 10       To State: 6       Prob: 100                    Times: 1

State:
S800             a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,(a*-d)>(b*-c) 0

State:
S801             a>b,c>d,-(a>d),-(-a+d),a*-d,-d>-c,b,-c,b*-c 0

State:
S802             a>b,c>d,-(a>d),(a*-d),-a+b
                 Action: 2        To State: 803     Prob: 100                    Times: 1

State:
S803             a>b,c>d,-(a>d),(a*-d),-a+b,b
                 Action: 30       To State: 804     Prob: 100                    Times: 1

State:
S804             a>b,c>d,-(a>d),(a*-d),-a+b,b,-c+d
```

```
                Action: 2           To State: 805     Prob: 100                        Times: 1

State:
S805            a>b,c>d,-(a>d),(a*-d),-a+b,b,-c+d,-c
                Action: 9           To State: 806     Prob: 66.6666666666667           Times: 2
                Action: 10          To State: 6       Prob: 33.3333333333333           Times: 1

State:
S806            a>b,c>d,-(a>d),(a*-d),-a+b,b,-c+d,-c,b*-c 0

State:
S807            a>b,c>d,-(a>d),--a+-d 0

State:
S808            a>b,c>d,-(a>d),a*-d,(a*c)>(b*d)
                Action: 4           To State: 809     Prob: 100                        Times: 1

State:
S809            a>b,c>d,-(a>d),a*-d,(a*c)>(b*d),-c
                Action: 3           To State: 810     Prob: 100                        Times: 1

State:
S810            a>b,c>d,-(a>d),a*-d,(a*c)>(b*d),-c,b
                Action: 10          To State: 6       Prob: 100                        Times: 1

State:
S811            a>b,c>d,-(a>d),-(c+-d)
                Action: 31          To State: 812     Prob: 100                        Times: 1

State:
S812            a>b,c>d,-(a>d),-(c+-d),-(c*-d)
                Action: 29          To State: 813     Prob: 100                        Times: 1

State:
S813            a>b,c>d,-(a>d),-(c+-d),-(c*-d),-d>-c
                Action: 1           To State: 814     Prob: 33.3333333333333           Times: 1
                Action: 9           To State: 814     Prob: 33.3333333333333           Times: 1
                Action: 31          To State: 815     Prob: 33.3333333333333           Times: 1

State:
S814            a>b,c>d,-(a>d),-(c+-d),-(c*-d),-d>-c,-(b>d) 0

State:
S815            a>b,c>d,-(a>d),-(c+-d),-(c*-d),-d>-c,(a*-d)
                Action: 30          To State: 816     Prob: 100                        Times: 1

State:
S816            a>b,c>d,-(a>d),-(c+-d),-(c*-d),-d>-c,(a*-d),-(-a+d)
                Action: 27          To State: 817     Prob: 100                        Times: 1

State:
S817            a>b,c>d,-(a>d),-(c+-d),-(c*-d),-d>-c,(a*-d),-(-a+d),a*-d
                Action: 9           To State: 818     Prob: 100                        Times: 1
```

```
State:
S818        a>b,c>d,-(a>d),-(c+-d),-(c*-d),-d>-c,(a*-d),-(-a+d),a*-d,(a>b)+(c>d) 0

State:
S819        a>b,c>d,-(a>d),a*-d,-c,b,(a+c)>(b+d)
            Action: 10          To State: 6       Prob: 100                 Times: 1

State:
S820        a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c+d,d+-c 0

State:
S821        a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c+d,-d>-c,-c
            Action: 10          To State: 6       Prob: 100                 Times: 1

State:
S822        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-d>-c
            Action: 1           To State: 823     Prob: 50                  Times: 1
            Action: 3           To State: 824     Prob: 50                  Times: 1

State:
S823        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-d>-c,a>-c 0

State:
S824        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-d>-c,b
            Action: 3           To State: 825     Prob: 100                 Times: 1

State:
S825        a>b,c>d,-(a>d),-(-a+d),a*-d,-c+d,-d>-c,b,-c
            Action: 10          To State: 6       Prob: 100                 Times: 1

State:
S826        a>b,c>d,-(a>d),-a+c 0

State:
S827        a>b,c>d,-(a>d),-a+b,-c+d,-a>-d

State:
S828        a>b,c>d,-(a>d),-(-a+d),a*-d,b,-d>-c,-c+d
            Action: 3           To State: 829     Prob: 100                 Times: 1

State:
S829        a>b,c>d,-(a>d),-(-a+d),a*-d,b,-d>-c,-c+d,-c
            Action: 10          To State: 6       Prob: 100                 Times: 1

State:
S830        a>b,c>d,-(a>d),-(-a+d),a*-d,b,-c+d,-c,b*-c 0
```

APPENDIX D: SAMPLE LOG DATA

Deep Thought Log File Sample. The log is generated on the web server where the instance of Deep Thought is running. The data includes the following columns:

Column 1: Student number (an encrypted id)
Column 2: Mode (English or Symbolic)
Column 3: Direction (Forward or Backwards)
Column 4: Rule (See Appendix B)
Column 5: Application (Contains old and new statements)
Column 6: Arg (The problem description)
Column 7: Level Number
Column 8: Problem Number
Column 9: Time on Step
Column 10: Server Time
Column 11: Server Date

student	mode	dir	rule	err	app	arg	lvl	prb	time	servDate	servTime
1001	eng	F	SIMP&	0	notCandA/A	S>(G&K),SvC,~C&A/G	1	1	0:01:48	0:01:48	10/5/2007
1001	eng	F	SIMP	0	notCandA/notC	S>(G&K),SvC,~C&A/G	1	1	0:00:10	0:01:58	10/5/2007
1001	eng	F	DS	0	SorC,notC/S	S>(G&K),SvC,~C&A/G	1	1	0:00:09	0:02:07	10/5/2007
1001	eng	F	MP	0	ifSthen(GandK),S/GandK	S>(G&K),SvC,~C&A/G	1	1	0:00:07	0:02:14	10/5/2007
1001	eng	F	SIMP~	0	GandK/G	S>(G&K),SvC,~C&A/G	1	1	0:00:06	0:02:20	10/5/2007
1001	sym	F	SIMP&	0	~I&Z/~I	G>(H&V),GvI,~I&Z/H	1	1	0:00:19	0:00:19	10/5/2007
1001	sym	F	DS	0	GvI,~I/G	G>(H&V),GvI,~I&Z/H	1	1	0:00:08	0:00:28	10/5/2007
1001	sym	F	MP	0	G>(H&V),G/H&V	G>(H&V),GvI,~I&Z/H	1	1	0:00:14	0:00:43	10/5/2007
1001	sym	F	SIMP	10		G>(H&V),GvI,~I&Z/H	1	1	0:00:07	0:00:50	10/5/2007
1001	sym	B	Extr	0	H	G>(H&V),GvI,~I&Z/H	1	1	0:00:02	0:00:53	10/5/2007
1001	sym	B	SIMP	0	H&q/H	G>(H&V),GvI,~I&Z/H	1	1	0:00:05	0:00:58	10/5/2007
1001	sym	B	Del	0	H&q	G>(H&V),GvI,~I&Z/H	1	1	0:00:07	0:01:05	10/5/2007
1001	sym	F	SIMP~	0	H&V/H	G>(H&V),GvI,~I&Z/H	1	1	0:00:06	0:01:12	10/5/2007
1001	sym	F	SIMP&	0	~B&W/~B	(OvU)>V,BvO,~B&W/V	1	2	0:00:15	0:00:15	10/5/2007
1001	sym	F	DS	0	BvO,~B/O	(OvU)>V,BvO,~B&W/V	1	2	0:00:08	0:00:23	10/5/2007
1001	sym	F	ADD	0	O/OvU	(OvU)>V,BvO,~B&W/V	1	2	0:00:16	0:00:40	10/5/2007
1001	sym	F	MP~	0	(OvU)>V,OvU/V	(OvU)>V,BvO,~B&W/V	1	2	0:00:08	0:00:49	10/5/2007
1001	sym	F	MT&	0	C>M,~M/~C	(~CvA)>(L&H),C>M,~M/H	1	3	0:00:08	0:00:08	10/5/2007
1001	sym	F	ADD	0	~C/~CvA	(~CvA)>(L&H),C>M,~M/H	1	3	0:00:16	0:00:25	10/5/2007
1001	sym	F	MP	0	(~CvA)>(L&H),~CvA/L&H	(~CvA)>(L&H),C>M,~M/H	1	3	0:00:05	0:00:31	10/5/2007
1001	sym	F	SIMP~	0	L&H/H	(~CvA)>(L&H),C>M,~M/H	1	3	0:00:07	0:00:38	10/5/2007

The NCSU log files show the final steps submitted for each student attempt. The header line of each logged attempt lists the student, problem, and whether the student completed the proof. A completed proof is listed as "Direct" and incomplete proofs are listed as "PartDirect". The log file has the following format:

Column 1: Step number
Column 2: Statement (a zero after the statement means the step was an error)

Column 3: Justification (Given or line numbers to justify)
Column 4: Reason (the rule or rules used. Rules are the actions seen in Appendix B)

Proof 1 stud: 81 prob: 1 PartDirect

	STATEMENTS		REASONS
1	a>b	GIVEN	
2	c>d	GIVEN	
3	-(a>d)	GIVEN	
4	d>a 0	3	29
4	-(-a+d)	3	30
5	a*d 0	4	27
5	a*-d	4	27
6	-d>-c	2	29
7	b 0	1, 5	3

Proof 1 stud: 161 prob: 2 Direct

	STATEMENTS		REASONS
1	a>b	GIVEN	
2	c>d	GIVEN	
3	-(a>d)	GIVEN	
4	--(a*-d)	3	31
5	a*-d	4	11
6	a	5	6
7	-d	5	6
8	b	6, 1	3
9	-c	7, 2	4
10	b*-c	8, 9	10

Proof 1 stud: 201 prob: 3 PartDirect

	STATEMENTS		REASONS
1	a>b	GIVEN	
2	c>d	GIVEN	
3	-(a>d)	GIVEN	
4	(a>c)*(b>d) 0	1, 2	8
4	-a*-d 0	3	28
4	-(a*-b)	1	32
5	-c-d 0	2	31
5	-(c*-d)	2	31
6	-d>-a 0	3	29
6	-a+b	4	28
7	-c+d	5	28

Proof 1 stud: 2 prob: 1 Direct

	STATEMENTS		REASONS
1	a>b	GIVEN	
2	c>d	GIVEN	

```
3   -(a>d)             GIVEN
4   -a+d 0              1     30
4   a*-d               3     27
5   a                4     6
6   b 0               3     3
6   b 0               3     3
6   b 0               1     3
6   b                1, 5   3
7   -d               4     6
8   -c               2, 7   30
9   b*-c              6, 8   10
```

Proof 1 stud: 5 prob: 2 Direct

```
          STATEMENTS               REASONS
1   a>b                GIVEN
2   c>d                GIVEN
3   -(a>d)             GIVEN
4   -(-a+d)             3     30
5   a*-d               4     27
6   -c+d               2     30
7   -d               5     6
8   -c 0              6, 7   2
8   d+-c 0              6     12
8   d+-c               6     12
9   -c 0              8     6
9   -c               7, 8   2
10  -a+b               1     30
11  a                5     6
12  b               10, 11 2
13  b*-c              9, 12  10
```

APPENDIX E: TERM DOCUMENT MATRIX

Example of Terms from term document matrix from Ch. 8 utility calculations.

Attempt	-(-d>-a)	-(d+-a)	a*-d	a>b	c>d	-(a>d)	-(-a+d)	-c
1			1	1	1	1		1
2			1	1	1	1	1	1
3			1	1	1	1	1	1
4			1	1	1	1	1	1
5			1	1	1	1		1
6			1	1	1	1	1	1
7			1	1	1	1		1
8			1	1	1	1	1	1
9			1	1	1	1	1	1
10				1	1	1		1
11			1	1	1	1	1	1
12			1	1	1	1	1	1
13			1	1	1	1	1	1
14			1	1	1	1	1	1
15			1	1	1	1	1	1
16			1	1	1	1	1	1
17			1	1	1	1	1	1
18	1	1	1	1	1	1		1
19	1	1	1	1	1	1		1
20			1	1	1	1	1	1
21			1	1	1	1		1
22				1	1	1		1
23			1	1	1	1	1	1
24				1	1	1		1
25			1	1	1	1	1	1
26			1	1	1	1	1	1
27			1	1	1	1		1
28			1	1	1	1	1	1
29			1	1	1	1	1	1
30				1	1	1		1
31			1	1	1	1	1	1
32			1	1	1	1	1	1
33				1	1	1	1	1
34	1		1	1	1	1		
35			1	1	1	1	1	1
36			1	1	1	1	1	1
37			1	1	1	1	1	1
38			1	1	1	1	1	1

39			1	1	1	1	1	1
40			1	1	1	1	1	1
41			1	1	1	1	1	1
42			1	1	1	1	1	1
43				1	1	1		1
44			1	1	1	1	1	1
45			1	1	1	1	1	
46			1	1	1	1	1	1
47				1	1	1		1
48			1	1	1	1	1	1
49			1	1	1	1	1	1
50			1	1	1	1	1	1
51			1	1	1	1	1	1
52			1	1	1	1	1	
53			1	1	1	1	1	1
54			1	1	1	1	1	1
55			1	1	1	1		1
56			1	1	1	1	1	1
57			1	1	1	1	1	1
58			1	1	1	1	1	1
59			1	1	1	1	1	1
60			1	1	1	1	1	1
61			1	1	1	1	1	1
62			1	1	1	1	1	1
63			1	1	1	1		1
64			1	1	1	1	1	1
65			1	1	1	1	1	1
66			1	1	1	1	1	1
67			1	1	1	1	1	1
68			1	1	1	1	1	1
69			1	1	1	1	1	1
70			1	1	1	1	1	1
71				1	1	1		1
72			1	1	1	1	1	1
73			1	1	1	1	1	1
74			1	1	1	1	1	1
75			1	1	1	1	1	1
76			1	1	1	1	1	1
77				1	1	1	1	
78			1	1	1	1	1	
79			1	1	1	1	1	
80			1	1	1	1		1
81			1	1	1	1		1
82			1	1	1	1		

83			1	1	1	1	1	1
84			1	1	1	1	1	1
85			1	1	1	1	1	1
86				1	1	1		1
87			1	1	1	1	1	1
88			1	1	1	1		1
89				1	1	1		1
90			1	1	1	1	1	1
91			1	1	1	1	1	1
92			1	1	1	1	1	1
93	1	1	1	1	1	1		1
94			1	1	1	1		
95			1	1	1	1	1	1
96			1	1	1	1	1	1
97			1	1	1	1	1	1
98			1	1	1	1	1	1
99			1	1	1	1	1	1
100				1	1	1		
101				1	1	1		1
102				1	1	1		1
103			1	1	1	1	1	1
104				1	1	1		1
105			1	1	1	1	1	1
106			1	1	1	1		1
107				1	1	1	1	1
108			1	1	1	1		1
109			1	1	1	1	1	1
110			1	1	1	1	1	1
111			1	1	1	1		1
112	1	1	1	1	1	1		1
113			1	1	1	1	1	1
114			1	1	1	1		1
115			1	1	1	1		1
116			1	1	1	1		1
117			1	1	1	1		1
118			1	1	1	1		1
119			1	1	1	1	1	1
120			1	1	1	1		1
121			1	1	1	1	1	1
122			1	1	1	1	1	1
123			1	1	1	1	1	1
124			1	1	1	1	1	1
125				1	1	1		1
126				1	1	1		1

127				1	1	1		
128			1	1	1	1	1	1
129				1	1	1		1
130			1	1	1	1	1	1
131				1	1	1		1
132				1	1	1		1
133			1	1	1	1	1	1
134			1	1	1	1	1	1
135			1	1	1	1	1	1
136			1	1	1	1	1	1
137			1	1	1	1	1	1
138			1	1	1	1	1	1
139			1	1	1	1	1	1
140			1	1	1	1	1	1
141			1	1	1	1	1	1
142			1	1	1	1	1	1
143			1	1	1	1	1	1
144			1	1	1	1	1	1
145			1	1	1	1	1	1
146			1	1	1	1	1	1
147			1	1	1	1	1	1
148			1	1	1	1	1	1
149			1	1	1	1	1	1
150			1	1	1	1	1	1
151				1	1	1		1
152			1	1	1	1	1	1
153			1	1	1	1	1	1
154			1	1	1	1	1	1
155			1	1	1	1	1	1
156			1	1	1	1	1	1
157			1	1	1	1	1	1
158			1	1	1	1		1
159			1	1	1	1	1	1
160			1	1	1	1	1	1
161				1	1	1		1
162			1	1	1	1	1	1
163				1	1	1	1	1
164			1	1	1	1	1	1
165			1	1	1	1		1
166			1	1	1	1	1	1
167			1	1	1	1	1	1
168			1	1	1	1	1	1
169			1	1	1	1	1	1
170			1	1	1	1	1	1

171			1	1	1	1	1	1
172			1	1	1	1		1
173				1	1	1		
174			1	1	1	1	1	1
175			1	1	1	1	1	1
176			1	1	1	1	1	1
177			1	1	1	1		1
178			1	1	1	1	1	1
179			1	1	1	1	1	1
180			1	1	1	1	1	1
181				1	1	1		1
182			1	1	1	1		1
183			1	1	1	1	1	1
184			1	1	1	1	1	1
185			1	1	1	1	1	1
186			1	1	1	1	1	1
187				1	1	1		1
188			1	1	1	1	1	1
189				1	1	1		1
190			1	1	1	1	1	1
191			1	1	1	1		
192				1	1	1		1
193			1	1	1	1	1	1
194			1	1	1	1	1	1
195			1	1	1	1	1	1
196			1	1	1	1	1	1
197			1	1	1	1	1	1
198	1	1	1	1	1	1		1
199				1	1	1		1
200			1	1	1	1	1	1
201			1	1	1	1		1
202			1	1	1	1	1	1
203			1	1	1	1	1	1
204			1	1	1	1	1	1
205			1	1	1	1		
206			1	1	1	1	1	1
207				1	1	1		1
208			1	1	1	1	1	1
209			1	1	1	1	1	1
210			1	1	1	1		1
211			1	1	1	1		
212			1	1	1	1	1	1
213			1	1	1	1	1	1
214			1	1	1	1		1

215			1	1	1	1		
216			1	1	1	1	1	1
217			1	1	1	1	1	1
218			1	1	1	1	1	1
219			1	1	1	1	1	1
220			1	1	1	1	1	1
221			1	1	1	1	1	1
222			1	1	1	1	1	1
223			1	1	1	1		1
224			1	1	1	1	1	1
225			1	1	1	1		1
226			1	1	1	1	1	1
227			1	1	1	1	1	1
228				1	1	1		1
229			1	1	1	1	1	1
230			1	1	1	1		1
231			1	1	1	1	1	1
232			1	1	1	1	1	1
233			1	1	1	1	1	1
234				1	1	1		1
235			1	1	1	1	1	1
236			1	1	1	1	1	1
237				1	1	1		1
238			1	1	1	1		
239			1	1	1	1	1	1
240			1	1	1	1		1
241			1	1	1	1	1	1
242			1	1	1	1	1	1
243			1	1	1	1	1	1
244			1	1	1	1	1	1
245				1	1	1		1
246			1	1	1	1	1	1
247			1	1	1	1	1	1
248			1	1	1	1	1	1
249			1	1	1	1	1	1
250			1	1	1	1	1	1
251	1		1	1	1	1	1	1
252				1	1	1		1
253			1	1	1	1	1	1
254			1	1	1	1		1
255			1	1	1	1	1	1
256			1	1	1	1	1	1
257			1	1	1	1		1
258			1	1	1	1		1

259			1	1	1	1	1	1
260			1	1	1	1	1	1
261			1	1	1	1	1	1
262			1	1	1	1	1	1
263			1	1	1	1	1	1
264			1	1	1	1	1	1
265			1	1	1	1	1	
266			1	1	1	1		
267			1	1	1	1	1	1
268			1	1	1	1		1
269			1	1	1	1	1	1
270			1	1	1	1	1	1
271			1	1	1	1	1	1
272			1	1	1	1	1	1
273			1	1	1	1		1
274			1	1	1	1	1	1
275			1	1	1	1	1	1
276			1	1	1	1		
277			1	1	1	1	1	1
278			1	1	1	1	1	1
279			1	1	1	1		1
280			1	1	1	1	1	1
281			1	1	1	1	1	1
282			1	1	1	1		1
283			1	1	1	1	1	1
284			1	1	1	1		1
285			1	1	1	1	1	1
286			1	1	1	1	1	1
287			1	1	1	1		1
288			1	1	1	1	1	1
289			1	1	1	1	1	
290			1	1	1	1	1	1
291			1	1	1	1	1	1
292			1	1	1	1	1	1
293				1	1	1		1
294			1	1	1	1	1	1
295			1	1	1	1	1	1
296			1	1	1	1	1	
297			1	1	1	1	1	1
298			1	1	1	1	1	1
299			1	1	1	1	1	1
300			1	1	1	1	1	1
301			1	1	1	1	1	1
302			1	1	1	1		1

303			1	1	1	1	1	1
304			1	1	1	1	1	1
305			1	1	1	1	1	1
306			1	1	1	1	1	1
307			1	1	1	1	1	1
308			1	1	1	1	1	1
309			1	1	1	1	1	1
310			1	1	1	1	1	1
311			1	1	1	1	1	1
312			1	1	1	1	1	1
313			1	1	1	1	1	1
314			1	1	1	1	1	1
315				1	1	1		1
316			1	1	1	1	1	1
317			1	1	1	1	1	1
318			1	1	1	1	1	1
319			1	1	1	1	1	1
320			1	1	1	1	1	1
321			1	1	1	1		1
322			1	1	1	1	1	1
323			1	1	1	1	1	1
324			1	1	1	1	1	1
325			1	1	1	1	1	1
326				1	1	1	1	1
327			1	1	1	1	1	1
328				1	1	1	1	1
329			1	1	1	1	1	1
330			1	1	1	1		1
331			1	1	1	1	1	1
332			1	1	1	1	1	1
333			1	1	1	1	1	
334			1	1	1	1	1	1
335			1	1	1	1	1	1
336			1	1	1	1	1	1
337			1	1	1	1	1	1
338			1	1	1	1	1	1
339			1	1	1	1	1	1
340			1	1	1	1	1	1
341			1	1	1	1	1	1
342			1	1	1	1	1	1
343			1	1	1	1	1	1
344			1	1	1	1		1
345				1	1	1		1
346			1	1	1	1	1	1

347				1	1	1		1
348			1	1	1	1	1	1
349				1	1	1		1
350			1	1	1	1	1	1
351			1	1	1	1	1	1
352			1	1	1	1		1
353			1	1	1	1	1	1
354			1	1	1	1	1	1
355			1	1	1	1	1	1
356			1	1	1	1		1
357			1	1	1	1	1	1
358			1	1	1	1	1	1
359			1	1	1	1	1	1
360			1	1	1	1	1	1
361			1	1	1	1	1	1
362			1	1	1	1	1	1
363			1	1	1	1	1	1
364			1	1	1	1	1	1
365			1	1	1	1	1	1
366			1	1	1	1	1	1
367	1	1		1	1	1		1
368				1	1	1		
369			1	1	1	1	1	1
370			1	1	1	1	1	1
371			1	1	1	1		1
372			1	1	1	1	1	1
373			1	1	1	1	1	1
374			1	1	1	1	1	
375			1	1	1	1	1	1
376			1	1	1	1	1	1
377				1	1	1		1
378			1	1	1	1		1
379			1	1	1	1	1	1
380				1	1	1		1
381			1	1	1	1	1	
Totals	8	6	333	381	381	381	276	356

APPENDIX F: EXAMPLE MODEL COMPONENTS

Level 1 MC computed from NCSU Problems.

Action	Stmts	New Stmt
ADD	-A+B	(-A+B)+A
ADD	-A	-A+B
ADD	A	A*(A>B)
ADD	A	A*-B
ADD	A	A+B
CD	(A>B)*(C>D)	(A*C)>(B*D)
CD	A>B,-C>-D	(A*-C)>(B*-D)
CD	A>B,C>D	(A*C)>(B*D)
CONJ	(A*(A>B))*-C,D>C	(B*-C)*(D>C)
CONJ	-A+B,(-A+B)*-B	(-A+B)*-B
CONJ	-A+B,(C*-B)*(-A*B)	(C*-B)*(-A*B)
CONJ	-A+B,(C*-B)*(-A+B)	(C*-B)*(-A+B)
CONJ	-A+B,-A+B	B*-C
CONJ	-A+B,-B	(-A+B)*-B
CONJ	-A+B,-B	-B*(-A+B)
CONJ	-A+B,-B	B*-B
CONJ	-A+B,-C+D	(-A+B)*(-C+D)
CONJ	-A+B,-C+D	B*-C
CONJ	-A+B,-C+D	B+-C
CONJ	-A+B,A	(-A+B)*A
CONJ	-A+B,A	A*(-A+B)
CONJ	-A+B,B*-C	B*-C
CONJ	-A,-A>-B	-A*(-A>-B)
CONJ	-A,-B+A	(-B+A)*-A
CONJ	-A,-B+A	-A*(-B+A)
CONJ	-A,-B	C*-B
CONJ	-A,B*-A	B*-A
CONJ	-A,B	-A*B
CONJ	-A,B	B*-A
CONJ	-A,B	B*-C
CONJ	-A,B>A	-A*(B>A)
CONJ	-A>-B,-A	(-A>-B)*-A
CONJ	-A>-B,-A	-A*(-A>-B)
CONJ	-A>-B,-B	C*-B
CONJ	A*-B,(-A+C)*(-D+B)	(A*-B)*((-A+C)*(-D+B))
CONJ	A*-B,(-A+C)*A	(-A+C)*A
CONJ	A*-B,-A+C	(A*-B)*(-A+C)
CONJ	A*-B,-B	A*-B
CONJ	A*-B,A	A*-B
CONJ	A*-B,A	C*-D
CONJ	A*-B,C	C*-B
CONJ	A+-B,-A	(A+-B)*-A
CONJ	A+-B,-A	-A*(A+-B)
CONJ	A,-A+B	(-A+B)*A

CONJ	A,-A+B	A*(-A+B)
CONJ	A,-B	-B*A
CONJ	A,-B	A*-B
CONJ	A,-B	C*-B
CONJ	A,-B	C*-D
CONJ	A,A*-B	A*-B
CONJ	A,A>B	A*(A>B)
CONJ	A,B*-C	B*-C
CONJ	A,B	B*-C
CONJ	A>B,-(A>C)	(A>B)*-(A>C)
CONJ	A>B,-B	(A>B)*-B
CONJ	A>B,A*-C	(A>B)*(A*-C)
CONJ	A>B,A	(A>B)*A
CONJ	A>B,A	A*(A>B)
CONJ	A>B,C+-D	B*-D
CONJ	A>B,C	(A>B)*-B
CONJ	A>B,C	C*(A>B)
CONJ	A>B,C>D	(A>B)*(C>D)
DEM	(A>B)*-(-A+C)	(A>B)*(A*-C)
DEM	(A>B)=(-A+B)	-(A*-B)
DEM	A+-B	-(-A*B)
DEM	A>B	-(A*-B)
DS	(-A+B)*-B	-A
DS	(-A+B)*A	B
DS	(A+-B)*-A	-B
DS	-A+B,-A	-A
DS	-A+B,-B	-A
DS	-A+B,A	B
DS	-A,-A*(A+-B)	-B
DS	-A,-B+A	-B
DS	-A,-B	-B
DS	-A,A+-B	-B
DS	A*(-A+B)	B
DS	A*-B	-B
DS	A+-B,-A	-B
DS	A+-B,B	A
DS	A,-A+B	B
DS	A,B	B
DS	A>B,-C>-D	(-C+A)>(-D+B)
DS	A>B,C>D	(A+C)>(B+D)
HS	A>B,-(-A+C)	B*-C
HS	A>B,C>D	B*-C
HS	A>B	-A+B
HS	A>B	-C
HS	A>B	B
HS	A>B	C*(-B>-A)
IMPL	(-A+B)*(-C+D)	(A>B)*(C>D)
IMPL	(-A+B)*A	(A>B)*A
IMPL	(A*B)>(C*D)	-(A*B)+(C*D)
IMPL	(A+B)>(C+D)	-(A+B)+(C+D)

IMPL	(A>B)*-(A>C)	(A>B)*-(-A+C)
IMPL	A*(-A+B)	A*(A>B)
IMPL	A>B,-B	-A
IMPL	A>B,C>D	B*-C
IMPL	A>B	(A>B)=(-A+B)
IMPL	A>B	-(A*-B)
IMPL	A>B	-A+B
MP	(-A>-B)*-A	-B
MP	(A*-B)>(C*-D),A*-B	C*-D
MP	(A>B)*A	B
MP	-A+B,-C	-D
MP	-A+B,A	B
MP	-A,-A>-B	-B
MP	-A>-B,-A	-B
MP	A*(A>B)	B
MP	A*-B,(A*-B)>(C*-D)	C*-D
MP	A*-B,A	C
MP	A+B,(A+B)>(C+D)	C+D
MP	A,A*(A>B)	B
MP	A,A>B	B
MP	A>B,-(A>C)	-(B>C)
MP	A>B,-(A>C)	B
MP	A>B,-(C>B)	-(C>A)
MP	A>B,-B	-A
MP	A>B,-C	B
MP	A>B,A	B
MP	A>B,B	B
MP	A>B	-(A+-B)
MT	(-A+B)>(-C+D),-(-C+D)	-(-A+B)
MT	(A>B)*-B	-A
MT	-A,B>A	-B
MT	-A>-B,--B	--A
MT	A*-B	-C
MT	A>B,(C*A)>(D*B)	-A
MT	A>B,-B	-A
MT	A>B	-A
MT	A>B	B
MT	A>B	C