An intelligent tutoring system for program semantics.

Steve Barker
Department of Computer Science, King's College, London

Paul Douglas
Cavendish School of Computer Science, University of Westminster

An Intelligent Tutoring System for Program Semantics

Steve Barker
King's College, London, UK
steve@dcs.kcl.ac.uk

Paul Douglas
University of Westminster, London, UK
P.Douglas@wmin.ac.uk

Abstract

In this paper, we describe an item of e-learning software that is intended to help students taking university computer science courses to understand the fundamentals of logic programming and deductive database semantics. The software is implemented in PROLOG and empowers students to explore their understanding of the semantics of logic programs and deductive databases. The software is also able to intelligently diagnose student misconceptions and includes a number of example programs/databases that permit students to test their understanding. We describe the development and evaluation of the software, and we present details of the analysis of the results of our investigation into the effectiveness of our e-learning tool. The results of our field study of the e-learning tool suggests that it of value in helping students to understand program and database semantics.

Keywords
E-Learning, Educational Software, Program Semantics.

1 Introduction

In this paper, we describe an e-learning tool that we have developed and used to help us to teach the declarative semantics of logic programs and deductive databases to undergraduate and postgraduate computer science students. The software empowers students to test hypotheses about a variety of logic semantics with respect to programs/databases that students select themselves to test their understanding of semantical issues. The software is able to diagnose a student's errors in understanding, it can explain errors of understanding, and it can suggest required corrections. Our e-learning tool is accessible via the Web and includes a front-end that users report to be motivating and useful to use for exploring aspect of program and database semantics.

Although a number of tools have been proposed in the literature on educational computing for helping students to learn aspects of logic (e.g., Tarski's World [4]), to the best of our knowledge, ours is the first piece of courseware that supports students in their learning about the declarative semantics of logic programs and deductive databases.

The rest of the paper is organized in the following way. Section 2 provides a brief introduction to declarative semantics of logic programs and deductive databases. In Section 3, some features of the software are described. In Section 4, the main results produced from the evaluation of the software are described and discussed. In Section 5, some conclusions are drawn, and suggestions are made for further work.

2 Semantics

In this section, we provide some background on program and database semantics to make the paper self-contained. For further details of program and database semantics we refer the reader to, e.g., [3].

The simplest form of program/database that we consider is definite programs/databases.

Definition 2.1 *A definite clause is a clause of the form:*

$$A \leftarrow B_1, \ldots, B_n.$$

In the definite clause $A \leftarrow B_1, \ldots, B_n$, A is called the *head* of the clause, and is an atomic formula; $B_1, \ldots, B_n$ comprises a conjunction of (positive) atomic formulae, and is called the *body* of the clause. If the body of $A \leftarrow B_1, \ldots, B_n$ is empty then $A \leftarrow$ is an *assertion* or a *fact*; otherwise $A \leftarrow B_1, \ldots, B_n$ is a deductive *rule*. In the case of a fact, $A \leftarrow$, the $\leftarrow$ is often dropped.

Definition 2.2 *A definite database is a finite set of definite clauses.*

Remark 2.1 *Our tool is based on the assumption that programs/databases are function-free.*

The semantics of a definite program/database is defined in terms of a least Herbrand model.

Definition 2.3 *The least Herbrand model $\mathcal{M}(\mathcal{D})$ of a $Datalog$ database $\mathcal{D}$ is the set of atoms in $HB(\mathcal{D})$ that are true in all Herbrand models of $\mathcal{D}$ (i.e., the set of atoms that are true in the intersection of all Herbrand models of $\mathcal{D}$).*[1]

Our e-learning tool enables the semantics of various *normal programs/databases* to be investigated.

[1]$\mathcal{D}$ axiomatizes $\mathcal{M}(\mathcal{D})$.

Definition 2.4 *A normal clause is a formula of the form:*

$$C \leftarrow A1, A2, \ldots, Am,$$
$$not\ B1, not\ B2, \ldots, not\ Bn.$$

Each $not\ Bj$ literal in the previous definition ($j \in \{1, .., n\}$) is a *negative literal*. In the case of a negative literal, the relevant type of negation is *negation as failure* [2].

Definition 2.5 *A normal database is a finite set of normal clauses.*

The semantics of normal programs that can be checked by our e-learning tool are the *stable model* and *well-founded* semantics.

Given the ground instance of a normal clause program/database G, $Ground(G)$, and a Herbrand interpretation I, the *Gelfond-Lifschitz transformation* [2] of G over I, $\mathcal{GL}(G, I)$, is formed by:

1. deleting from $Ground(G)$ each clause with a negative condition in the body, $not\ A$, where A is in I (i.e., A is true in I).

2. deleting each negative condition $not\ A$ from the remaining clauses of $Ground(G)$ if A is not in I (i.e., A is false in I).

Definition 2.6 (Stable Model Semantics) *$\mathcal{GL}(G, I)$ transforms a normal program/database G into a definite program/database D with a least H-model, M_D. I is a stable model of G iff $I = M_D$.*

There are various representations of the well founded semantics. We consider the definition due to Przymusinska and Przymusinski [8]. This approach is based upon a 3-valued generalisation of the Gelfond-Lifschitz method for computing 2-valued stable models.

The Przymusinski-Przymusinska approach makes use of the notion of a *partial interpretation*, $I = \langle\{T\}, \{F\}\rangle$, where T is the set of atoms that are true in I, and F is the set of atoms that are false in I. All atoms in $HB(G) - (T \cup F)$ have an undefined truth value (where $-$ is the set difference operator).

The partial interpretation I is used with the following three-step procedure for transforming the ground instance of a normal clause program/database G, $Ground(G)$, into a non-negative program/database ΔG:

- for any atom $A \in I$, if A is true in I and $B \leftarrow not\ A \in Ground(G)$ then remove this clause from $Ground(G)$ to give the program/database $G1$.
- replace, in all of the clauses in $G1$, any $not\ A$ condition with u (where u denotes the unknown truth value) if A is an atom with an undefined truth value in I. This generates the program/database $G2$.
- for any atom $A \in I$, if A is false in I and $B \leftarrow not\ A \in G2$ then remove the $not\ A$ condition from all such clauses in $G2$ to give the program/database ΔG.

As ΔG is non-negative it follows that ΔG has a least 3-valued model, L (say). If $I = L$ then I is a 3-valued stable model of G. Moreover, we have the following result.

Theorem 2.1 *If I is a 3-valued stable model of the program/database G that is generated by the Przymusinski-Przymusinska transformation then I is the smallest (i.e., most sceptical) 3-valued stable model of G, and also the well-founded model of G [8].*

3 The E-Learning Tool: Development and Features

Our e-learning tool is written in PROLOG [1]. PROLOG has been widely used for implementing items of educational software (see, for example, [9] and [7]) and is appropriate for developing applications, like ours, which require that some form of "intelligence" be captured. The fact that the programs/databases that the software processes can be directly translated into PROLOG's rule-based language was another reason for choosing the latter for the implementation of the e-learning tool.

The first stage in developing our software involved us using a *phenomenographic* method [6] for information gathering on students' understanding of concepts of logic programs, deductive databases and semantics. By conducting 'dialogue' sessions with students we identified the way students seek to understand the semantics of programs and databases. From our review of the notes taken at the dialogue sessions, we were able to develop a prototype system for facilitating student understanding of semantical issues.

Our design of the e-learning tool has been influenced by Gagne's work [5]. Gagne's event-based model of instruction helped us to decide what an individual learner ought to be offered and the order in which information ought to be presented to them. Following Gagne's suggestions, when students use the e-learning tool they are reminded what the learning task to be performed is, and what it is they are supposed to be able to do once the learning task has been completed. Prominence is given to the distinctive features that need to be learned, different levels of learning guidance are supported for different types of learners, informative feedback is given, and learning takes place in a student-centered, interactive way, but with support available to students as and when they need it.

The input to our e-learning tool is a propositional program/database $\mathcal{D}$ together with the model of the program/database that the student hypothesises as holding. The user selects a semantics to evaluate with respect to $\mathcal{D}$. In the remainder of this section we describe some examples showing how the system might be used.

Example 3.1 *Consider the program/database:*
$G_1 = \{p \leftarrow not\ q, r;\ r \leftarrow\}$

and the interpretation:

$I = \{p, r\}$

which is tested with respect to the stable model semantics.

The Gelfond-Lifschitz transformation of G_1 *over I produces the definite program,* D_1*=*$\{p \leftarrow r;\ r \leftarrow\}$*, which has the least H-model {p,r}. As {p,r} is the least H-model of* D_1 *it follows that {p,r} is a stable model of* G_1*. The program outputs this result together with the intermediate results showing how it was obtained.* $\diamond$

Example 3.2 *Consider the program/database:*

$G = \{c \leftarrow not\ d; a \leftarrow not\ b; b \leftarrow not\ a\}$

and the interpretation:

$I = \langle\{c\}, \{d\}\rangle.$

which is tested with respect to the well founded semantics.

The transformation of G into its non-negative form with respect to I gives: $G3 = \{c \leftarrow; a \leftarrow u; b \leftarrow u\}$*. As the least model of* $G3$ *coincides with I, I is the well-founded model of G.*[2] $\diamond$

4 Evaluation of the E-learning Tool

We have performed a formative evaluation and a summative evaluation of our e-learning tool.

In brief, the aim of the formative evaluation of the e-learning tool was to provide information that would enable us to develop the software to a point at which it could be summatively evaluated. The summative evaluation was intended to help us to decide whether the e-learning tool was of value in helping students to understand the details of program/database semantic satisfaction; how the e-learning tool compared in this respect to texts on program/database semantic satisfaction; and the extent to which each means of instruction was perceived by students to be motivating to use (or otherwise), and of value in helping them to learn about the declarative semantics of logic programs.

[2]To preserve the law of identity as a theorem, $u \leftarrow u$ is true in Lukesiewicz's 3-valued logic, and Kleene's weak interpretation of $\leftarrow$.

Although our study was primarily concerned with comparing the e-learning tool with existing work on declarative semantics, it should be noted that we do not envisage that the two modes of instruction should be used in a mutually exclusive way. The comparison of the e-learning tool with texts on the semantics of programs/databases in our evaluation was chosen merely to attempt to decide whether there was any evidence to suggest that the former might have some "educational value" when compared to the latter.

For the formative evaluation, comments on the e-learning tool were sought from: two members of the teaching staff at the University of Westminster (the "expert reviewers"); a volunteer student from the university's MSc course in Database Systems (the one-to-one study); and a group of six volunteer students from the same course (the small-group testing). The volunteer students were randomly allocated to either the one-to-one or small-group testing (but not both). These students were learning about declarative semantics of logic programs at the time at which the formative evaluation of the e-learning tool was being conducted.

In response to the feedback received from the users of the software, a number of changes were made to the e-learning tool over time. For example, the explanations were changed to make them less technical and less formal that their original form, and a variety of modifications were made to the front-end to enhance its appeal. The power to investigate any program/database and any semantics was reported to be an attraction of the e-learning tool, and a major advantage it had over the texts that we used to support the delivery of material. The students commented that they particularly liked the fact that they could control the pace and delivery of learning material and could investigate issues of their own choosing.

The software was summatively evaluated with 21 students at the University of Westminster who were taking courses in logic programming and deductive databases in the Second Semester of the 2003/04 academic year.

A 5-point Likert scale, with 24 statements, was used to collect data about the perceptions the students had of the e-learning tool as a method for facilitating understanding of declarative semantics of logic programs and deductive databases, and its attractiveness as a learning instrument. To analyse the data produced from the Likert scale, we chose to use a t-test; the idea was to compare the matched pairs of scores produced by each respondent for the e-learning tool relative to the course notes that were provided on semantics and the books that students had read on the semantics of programs/databases. Henceforth, we use $\mathcal{T}$ to denote the textual sources that were used for comparison with our e-learning tool.

To analyze the information produced from the Likert scale, t-statistics were computed to compare the mean scores for the perceptions students had of the e-learning tool and $\mathcal{T}$, overall and for three specific measures: perceived helpfulness as a teaching aid, motivational appeal, and the value of the explanation and diagnostic modules.

The results produced from the Likert scale were very clear. In the overall measure of the two methods, the average difference in the ratings of the software and $\mathcal{T}$ was 15.92 in favour of the e-learning tool, and only two students reported that $\mathcal{T}$ was "better" than the e-learning tool. The t-statistic for the comparison of average differences was 6.85. This is statistically significant at the 2% level.

Not surprisingly, given the overall results, the e-learning tool was also perceived to be "better" than $\mathcal{T}$ in all three of the sub-categories of Likert scale items.

In terms of helping students to understand declarative semantics of logic programs/deductive databases, the average difference in scores between the e-learning tool and $\mathcal{T}$ was 2.02, in favour of the e-learning tool, and all but three of the students reported that the e-learning tool had been of more value than $\mathcal{T}$ for helping them to learn about logical semantics. In the t-test comparison of the average difference in the ratings of the e-learning tool and $\mathcal{T}$, the t-statistic was 3.59. This value is significant at the 2% level.

Our software was also perceived to have more motivational appeal than $\mathcal{T}$. The average difference in the rating of the e-learning tool and $\mathcal{T}$ in this case was 10.27 and every student reported that the e-learning tool had been more motivating to use than $\mathcal{T}$. The t-value of 7.39 for the comparison of average differences in ratings between the e-learning tool and $\mathcal{T}$ is significant at the 1% level.

The explanations provided by the e-learning tool were unanimously perceived by the students to be of more value than those in $\mathcal{T}$ for helping them to understand the declarative semantics of logic programs/deductive databases. The average difference in scores on the value of the exercises, explanations was 4.57 in favour of the e-learning tool. The t-statistic for the average difference was 8.87 which is (again) significant at the 1% level. The diagnostic components were reported to be helpful in correcting misunderstanding and a positive feature that was not shared by $\mathcal{T}$.

5 Conclusions and Further Work

Our software shows that a suitable e-learning tool can be developed to help computer science students to learn about the declarative semantics of logic programs and deductive databases. The e-learning tool enables students to construct their own learning environments and is able to interpret and explain a student's mistakes as well as being able to confirm it when his/her understanding is correct. As such, the e-learning tool provides students with "intelligent" tutorial support for learning about the declarative semantics of logic programs and deductive databases. The tool is based on sound principles of learning, it is able to deal with any syntactically correct program/database as input, and it can be extended to accommodate any number of examples or exercises without requiring changes to the core set of rules on which the e-learning tool is based.

Using the e-learning tool enables students to: choose to investigate various declarative semantics by using inputs of their own choice; make hypotheses about programs/databases satisfying declarative semantics; test these hypotheses; and explore the consequences of program/database semantics satisfaction by changing the inputs. As such, the e-learning tool empowers students to take control of their own learning, they can learn at their own preferred pace, they can investigate their own misunderstandings and reinforce their own understanding of the declarative semantics of logic programs and deductive databases. The fact that the software encourages students to "learn by hypothesizing" is particularly important because this is the approach students naturally adopt to learn about declarative semantics of logic programs. Using textbooks does not enable this type of learning to be supported, and can only offer students a limited number of examples of logical semantics property satisfaction; textbooks cannot provide interactive feedback to students investigating semantics and semantics of their own choosing. Unlike their human tutors, the e-learning tool has the additional attraction of providing students with tutorial support in their learning of logical semantics properties whenever they require it (via the Internet).

The results produced by our summative assessment of the e-learning tool indicate that it was perceived by our students to be superior to $\mathcal{T}$ in a number of respects. However, more work will be required on the issue of student perceptions of the e-learning tool and $\mathcal{T}$ before any definite conclusions may be drawn about their relative value. Our experience of conducting this study has also revealed that some students appear to favour e-learning tools over traditional sources of learning simply because of the novelty of the former. An investigation of the implications of this attitude is a matter for future study. Moreover, while the Likert scale test revealed that the e-learning tool was perceived to be helpful to students learning about declarative semantics of logic programs, further research is required to try to establish *why* exactly this is the case (e.g., to what extent are student perceptions influenced by the novelty value of e-learning tools?).

A number of extensions to the e-learning tool are possible. For example, it could be extended to permit syntactic properties to be evaluated (e.g., *stratification* or *local stratification*), it may be extended to incorporate other semantics (e.g., *answer sets*) for other classes of programs/databases (e.g., *disjunctive databases*), and the tool may be extended to permit first order programs/databases to be checked for the semantical properties that they satisfy.

References

[1] I. Bratko. *PROLOG Programming for Artificial Intelligence*. Addison-Wesley, 1986.

[2] K. L. Clark. *Negation As Failure*, pages 293–322. Plenum Press, 1972.

[3] S. K. Das. *Deductive Databases and Logic Programming*. Addison-Wesley, 1992.

[4] S. U. C. for the Study of Language and Information. *Tarski's World*. http://www.-csli.stanford.edu/hp/.

[5] R. M. Gagne. *The Conditions of Learning*. Holt, Reinhart and Winston, 1970.

[6] F. Marton and P. Ramsden. *What does it take to improve learning?* Kogan Page, 1988.

[7] J. Nichol, J. Briggs, and J. Dean. *Prolog, Children and Students*. Kogan-Page, 1988.

[8] H. Przymusinska and T. C. Przymusinski. Weakly perfect model semantics for logic programs. In *Proceedings of the 5th International Conference and Symposium on Logic Programming*, pages 270–279, Seattle, USA, 1988.

[9] M. Yazdani. *New Horizons in Educational Computing*. Ellis Horwood, 1983.