A Database Transaction Scheduling Tool in Prolog

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 "intelligent" educational software that is intended to help students taking university computer science courses to understand the fundamentals of transaction scheduling. The software, implemented in PROLOG, empowers students to construct their own learning environment and is able to provide tailored forms of feedback to different types of learner. 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 the software as a teaching and learning tool. Our results suggest that our learning tool provides students with a different and valuable type of learning experience, which traditional methods do not provide.

1 Introduction

In this paper, we describe an item of educational software that we have developed and used to help us to teach certain key notions from the realms of database transaction processing to undergraduate computer science students. More specifically, the software is an educational tool that is intended to "intelligently" assist computer science students in developing their understanding of CRAS property satisfaction [1]. In this context, "intelligently" may be interpreted as an ability to respond to a student's self-selected input by detecting and explaining his/her errors to them or confirming that his/her understanding is correct.

Ours is one of the first pieces of courseware to provide students with help in understanding the basic notions of database transaction processing and, to the best of our knowledge, is the first piece of software that is specifically intended for helping students to learn about the CRAS properties. The software provides students with a tutorial aid that is able to respond to questions about CRAS

property satisfaction in the same way that an "expert tutor" might; it enables a student to investigate the CRAS properties at his/her leisure and enables teaching staff to use tutorial sessions to answer any "non-standard" questions that students might have. This tool is also important because it provides students with a learning experience that no textbook can provide. More specifically, the software encourages students to learn about the CRAS properties by making and testing hypotheses. This approach appears to be the most natural way for students to learn about the CRAS properties (certainly it is the approach they naturally adopt). The traditional, text-based method that we have previously used to teach material on CRAS property satisfaction does not support learning by hypothesis formulation.

The rest of the paper is organized in the following way. Section 2 provides a brief introduction to the CRAS properties. In Section 3, some key features of the software are outlined. 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.

We assume that the reader has a basic knowledge of database transaction processing; otherwise, we suggest [2] for introductory material.

2 The CRAS Properties

The CRAS properties are criteria that should be satisfied by a *schedule*.

Definition 2.1 A schedule σ over a set of transactions $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ is a strict partial order $(\mathcal{T}, <)$ (where $<$ is an "earlier than" relation) with the following properties (where o_i is an operation performed by transaction t_i and the allowed operations are r (read) and w (write)):

1. $\forall o_i \in \sigma$ (where $o \in \{r, w\}$), $\exists t_i \in \mathcal{T}$ such that $o_i \in t_i$;

2. If o_i and o_j are operations in $t_i \in \mathcal{T}$ and $o_i < o_j$ holds in t_i then $o_i < o_j$ holds in σ;

3. For any two conflicting operations o_i and o_j in σ, $o_i < o_j$ xor $o_j < o_i$.

Remark 2.1 Our definition of a schedule implies that no duplicate operations on a data item are possible, and assumes that a single CPU is used in transaction processing. If parallel processing is possible then $<$ can be replaced by $\leq$ in Definition 2.1.

Unfortunately, certain interleavings of the operations from different transactions in a schedule can cause anomalous behaviours (which cause the integrity of the data in a database to be compromised) and can raise a number of practical difficulties. For instance, it is possible for the value of a data item which has been updated by one transaction t_i in a schedule to be overwritten by another transaction t_j before t_i's update is performed on the database (this phenomenon is usually referred to as the *lost update problem* [2]). The principal sources of problems that arise when concurrently processing transactions are *conflicting operations* and *read froms* that involve reading uncommitted data.

Definition 2.2 Two operations o_i and o_j in a schedule σ are a conflicting pair (or are in conflict) iff the following conditions hold:

- o_i and o_j are operations on a common data item;
- at least one of o_i and o_j is a write operation.

Definition 2.3 For each data item x, if (i) $w_i(x) < r_j(x)$ holds in a schedule, (ii) t_i does not abort before $r_j(x)$, and (iii) every transaction (if any) that writes x between $w_i(x)$ and $r_j(x)$ aborts before $r_j(x)$ then t_j reads from t_i.

The CRAS properties help to solve certain problems, that arise as a consequence of interleaving of operations, by imposing certain constraints on the order in which operations are performed in a schedule. By ensuring that these constraints are satisfied, a database management system is guaranteed to produce schedules that are free of a class of potential problems that may violate the integrity of the data contained in a database. Moreover, satisfaction of the CRAS properties permits a DBMS to be configured to optimize the performance of transaction management.

The CRAS properties are: conflict serializability, recoverability, avoids cascading aborts and strictness. These properties are defined formally below. In these definitions, t_i and t_j denote arbitrary transactions, $T(\sigma)$ denotes an arbitrary schedule σ defined on a set of transactions T, r_i, w_i, a_i and c_i are respectively read, write, abort and commit operations by transaction t_i, $\rightarrow$ is "implication", $\wedge$ is 'and', $\vee$ is 'or', $\neg$ is negation, and $<$ denotes the "earlier than" relationship between operations.

Definition 2.4 A schedule σ on a set of transactions T is conflict serializable iff the following holds:

$$\forall t_i, t_j \in T(\sigma)\ conflict(t_i, t_j) \rightarrow \neg conflict(t_j, t_i)$$

where conflict is defined thus:

$$\forall t_i, t_j \in T(\sigma)\ r_i(x) < w_j(x) \rightarrow conflict(t_i, t_j)$$

$$\forall t_i, t_j \in T(\sigma)\ w_j(x) < r_i(x) \rightarrow conflict(t_j, t_i)$$

$$\forall t_i, t_j \in T(\sigma)\ w_i(x) < w_j(x) \rightarrow conflict(t_i, t_j).$$

Definition 2.5 A schedule σ on a set of transactions T is recoverable iff the following holds:

$$\forall t_i, t_j \in T(\sigma)\ read\_from(t_i, t_j) \rightarrow c_j \in \sigma\ \wedge\ c_j < c_i.$$

Definition 2.6 A schedule σ on a set of transactions T avoids cascading aborts iff the following holds:

$$\forall t_i, t_j \in T(\sigma)\ read\_from(t_i, t_j) \rightarrow c_j < r_i(x) \vee a_j < r_i(x).$$

Definition 2.7 A schedule σ on a set of transactions T is strict iff the following holds:

$$\begin{aligned}\forall t_i, t_j \in T(\sigma)\ w_j(x) < r_i(x) \vee w_j(x) < w_i(x) \rightarrow \\ a_j < r_i(x) \vee c_j < r_i(x) \vee \\ a_j < w_i(x) \vee c_j < w_i(x).\end{aligned}$$

Definition 2.8 The auxiliary predicate read_from is defined thus:

$$\begin{aligned}\forall t_i, t_j \in T(\sigma)\ \exists x[read\_from(t_i, t_j) \leftarrow\ w_j(x) < r_i(x) \wedge \neg(a_j < r_i(x)) \wedge \\ [\forall t_k \in T(\sigma)\ w_j(x) < w_k(x) < r_i(x) \\ \rightarrow a_k < r_i(x)]].\end{aligned}$$

Conflict serializability is the principal schedule correctness criterion that is used in practice by DBMS to avoid problems of inconsistent updating that arise during concurrent transaction execution. The recoverability criterion must be satisfied in order to preserve the semantics of commit operations in schedules. The ACA condition is a practical criterion that, if satisfied, reduces the amount of work that needs to be performed to recover from the effects of failed transactions. As the name suggests, satisfying the avoiding cascading aborts condition ensures that if a transaction aborts then it does not cause a chain (or cascade) of aborting

transactions to arise. That is, the failure of one transaction, t_1 (say), does not cause another transaction, t_2 (say), to fail which causes another transaction t_3 (say) to fail and so on. Strictness enables a particularly efficient method to be employed to manage aborted transactions i.e., by reinstalling *before images*.

3 Some Key Features of the Software

Our teaching tool enables students to test any syntactically correct schedule that they choose as input to the system. Students also have complete freedom to choose to investigate the satisfaction of any of the CRAS properties by these schedules.

The software that implements the system is written in PROLOG [3]. PROLOG has been widely used for implementing items of educational software (see, for example, [4] and [5]) and is appropriate for developing applications, like ours, which require that some form of "intelligence" be captured. The fact that the rules that define the CRAS properties can be directly translated into PROLOG's rule-based language was another reason for choosing the latter for the implementation of the software.

Our design of the software has been influenced by Gagne's work [6]. 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 software 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-centred, interactive way, but with support available to students as and when they need it.

When engaging with the software, a user enters a schedule and selects a CRAS property to evaluate with respect to the schedule. The schedule is displayed to the user who may then pose queries on the schedule to test it for satisfaction of CRAS properties with respect to the set of axioms $\mathcal{A}$ that defines these properties and the auxiliary predicates in terms of which the CRAS properties are defined (see Definitions 2.4-2.8). The axioms in $\mathcal{A}$ are converted into PROLOG code for implementation.

Each operation in a schedule may be represented by a 4-tuple, (o,t_j,i,t_s). Here, o denotes an operation (i.e. read or write), t_j denotes a transaction performing the operation, i denotes the data item read or written by t_j, and t_s is the time at which o is performed i.e., the timestamp for o. In the case where o is a commit

or an abort, the data item is *null* since these operations are not performed on a data item. In our PROLOG implementation, each 4-tuple that describes an operation is represented as a fact of the form $o(a, t_j, i, t_s)$ where $a \in \{r, w\}$. In this context, a schedule σ is a finite set of o operations, and a PROLOG program is a pair $(\mathcal{A}^P, \sigma)$ where $\mathcal{A}^P$ is the PROLOG form of $\mathcal{A}$.

Example 3.1 The representation of a conflicting pair (N, M) of transactions may be expressed in PROLOG, thus (cf. Definition 2.2):

$$\begin{aligned} conflict(N, M) : -o(r, N, Y, T1), \\ o(w, M, Y, T2), \\ T1 < T2, N = \backslash = M. \end{aligned}$$

$$\begin{aligned} conflict(M, N) : -o(r, N, Y, T1), \\ o(w, M, Y, T2), \\ T2 < T1, N = \backslash = M. \end{aligned}$$

$$\begin{aligned} conflict(N, M) : -o(w, N, Y, T1), \\ o(w, M, Y, T2), \\ T1 < T2, N = \backslash = M. \end{aligned}$$

An example of the output for a schedule σ produced in a user session and an example of engaging with the system follows next.

Example 3.2 Suppose that a user's choice of schedule is as follows:

$$\langle w_1(x), w_1(y), r_2(u), w_1(z), w_2(z), c_1, w_2(x), r_2(y), w_2(y), c_2 \rangle$$

Then, the software displays the user schedule, thus:

Your chosen schedule was:

w,1,x,90
w,1,y,95
r,2,u,100
w,1,z,105
w,2,z,110

c,1,null,115
w,2,x,120
r,2,y,125
w,2,y,130
c,2,null,135

Thereafter, the user may evaluate CRAS properties with respect to the schedule. For example, the user may ask "is this schedule an ACA schedule?" i.e., *?-aca.*. In this case the output is "*yes*". A user can ask for an explanation of this result by posing the following query: *?-explainaca.* To which the software will respond:

> *Transactions in this schedule only read data items AFTER they have been written by transactions that have committed.*

Similarly, the query *?-st.* for the schedule above (i.e. "is this schedule strict?") is answered "*no*" by the software. If the user then poses the query *?-explainnonst.* (i.e. why is this schedule not strict?) then the software will respond with the following explanation:

> *Transaction 2 overwrites the data item z written by transaction 1 but BEFORE transaction 1 reaches its commit point.*

All CRAS property satisfaction questions are evaluated as described in the previous example, and several levels of explanation are provided by the software.[1]

4 Evaluation of the Software

We have performed a formative evaluation and a summative evaluation of our teaching tool.

In brief, the aim of the formative evaluation of the software 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 software was of value in helping students to understand the details of CRAS property satisfaction; how the software compared in this respect to the standard text on CRAS property satisfaction [7]; 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 CRAS properties.

[1] Several levels of explanation are offered in the sense that users are able to check the correctness of a query in respect of any of the CRAS properties. If the query is not correct they can attempt to correct it; alternatively, they can ask the software *why* it is not correct.

Although our study was primarily concerned with comparing the software with [7], 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 software and [7] 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 software 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 four 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 the CRAS properties at the time at which the formative evaluation of the software was being conducted.

In response to the feedback received from the users of the software, a number of changes were made to the software over time. For example, the software was changed from its initial form to display a user's schedule together with the explanations of the CRAS property satisfaction or violation by a schedule, and a variety of modifications were made to the front-end to enhance its appeal. The power to investigate any schedule and CRAS property was reported to be an attraction of the software, and a major advantage it had over Bernstein et al's text. The students commented that they particularly liked the fact that they could "interact" with the software, and that it "lets you decide what to learn".

The software was summatively evaluated with the cohort of 27 students at the University of Westminster who were taking the Database Administration (DBA) module as part of their BSc Computer Science degree programme in the Second Semester of the 2002/03 academic year.

A 5-point Likert scale, with 22 statements, was used to collect data about the perceptions the students had of the software and [7] as methods for facilitating understanding of the CRAS properties, and their attractiveness as learning instruments. 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 software and [7].

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 software and [7], overall and for three specific measures: perceived helpfulness as a teaching aid, motivational appeal, and the value of the on-line exercises, examples, and explanations.

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 [7] was 17.24 in favour of the software, and only one student reported that [7] was "better" than the software. The t-statistic for the comparison of average differences was 7.75. This is statistically significant at the 1% level.

Not surprisingly, given the overall results, the software was also perceived to be "better" than [7] in all three of the sub-categories of Likert scale items.

In terms of helping students to understand the CRAS properties, the average difference in scores between the software and [7] was 2.18, in favour of the software, and all but two of the students reported that the software had been of more value than [7] for helping them to learn about the CRAS properties. In the t-test comparison of the average difference in the ratings of the software and [7], the t-statistic was 3.48. This value is significant at the 2% level.

Our software was also perceived to have more motivational appeal than [7]. The average difference in the rating of the software and [7] in this case was 10.65 and every student reported that the software had been more motivating to use than [7]. The t-value of 7.53 for the comparison of average differences in ratings between the software and Bernstein et al. is significant at the 1% level.

The exercises, explanations and examples that are included in the software were almost unanimously perceived by the students to be of more value than those in [7] for helping them to understand the CRAS properties (for one student, however, the difference in their scores for the software and [7] was zero). The average difference in scores on the value of the exercises, explanations and examples was 4.41 in favour of the software. The t-statistic for the average difference was 8.45 which is (again) significant at the 1% level.

5 Conclusions and Further Work

Our software shows that a suitable tool can be developed to help computer science students to learn about the CRAS properties. The package enables students to construct their own learning environments by using a piece of courseware that is able to interpret and immediately explain a student's mistakes as well as being able to confirm it when his/her understanding is correct. As such, the software provides students with "intelligent" tutorial support for learning about the CRAS properties. The software is also based on sound principles of learning [7], is able to deal with any syntactically correct schedule, 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 software is based.

Using the software enables students to: choose to investigate any of the CRAS properties using schedules of their own choice; make hypotheses about schedules satisfying the CRAS properties; test these hypotheses; and explore the conse-

quences of CRAS property satisfaction by manipulating the operations included in a schedule. As such, the software 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 CRAS properties. 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 the CRAS properties. Using textbooks does not enable this type of learning to be supported, and can only offer students a limited number of schedules and examples of CRAS property satisfaction; textbooks cannot provide interactive feedback to students investigating schedules and schedule properties of their own choosing. Unlike their human tutors, the software has the additional attraction of providing students with tutorial support in their learning of the CRAS properties whenever they require it.

The results produced by our summative assessment of the software indicate that it was perceived by our students to be superior to [7] in a number of respects. However, more work will be required on the issue of student perceptions of the software and [7] before any definite conclusions may be drawn about their relative value. Our experience of conducting this study has also revealed that some students have a tendency to believe that a piece of educational software has to invariably be better than a text; these students regard the former as being "the future" whilst the latter is viewed as being distinctly *passé*. We intend to investigate the implications of this attitude in the near future. Moreover, while the Likert scale test revealed that the software was perceived to be helpful to students learning about the CRAS properties, further research is required to try to establish *why* this is the case.

A number of extensions to the software are possible. For example, it could be extended to permit the investigation of other types of schedule properties (e.g. *rigour* [8]) and with minor modifications the software can be used as a tutorial aid for learning about *optimistic concurrency control* [9]. We would also like to investigate the addition of a more "friendly" user environment.

6 References

[1] Barker, S., Proving Properties of Schedules, Proc. IEEE Workshop on Knowledge and Data Engineering, 174-180, 1998.

[2] Gray, J. and Reuter, A. (1993) *Transaction Processing: Concepts and Techniques*, San Mateo, CA: Morgan Kaufmann.

[3] Bratko, I. (1986) *PROLOG Programming for Artificial Intelligence*, Reading, MA: Addison-Wesley.

[4] Yazdani, M. (1983) *New Horizons in Educational Computing*, Chichester: Ellis Horwood.

[5] Nichol, J., Briggs, J., and Dean, J. (1988) *Prolog, Children and Students*, London: Kogan-Page.

[6] Gagne, R. M. (1970) *The Conditions of Learning*, NY: Holt, Reinhart and Winston.

[7] Bernstein, P., Goodman, N., and Hadzilacos, V. (1987) *Concurrency Control and Recovery in Database Systems*, Menlo Park, CA: Addison Wesley.

[8] Briebart, Y., Georgakopoulos, D., Rusinkiewicz, M., and Silbershatz, A. (1991) On Rigorous Transaction Scheduling, *IEEE Transactions on Software Engineering*, 17, 954-960.

[9] El-Masri, R. and Navathe, S. (2003) *Fundamentals of Database Systems*, Redwood City, CA: Benjamin Cummings.