Empirical Modelling in support of constructionism: a case study

Meurig Beynon, Antony Harfield
Department of Computer Science, University of Warwick, Coventry CV4 7AL
{wmb,ant}@dcs.warwick.ac.uk

Abstract

Conventional programming paradigms have limitations where support for constructionist learning is concerned. This paper illustrates the merits of an alternative approach to giving support for constructionist learning, based on the principles of Empirical Modelling (EM), with reference to an algorithm for determining whether a decomposition of a particular relational schema is lossless. Model-building that is to be effective for constructionist learning has to support activities relating to three distinct roles: that of a student, a teacher and a developer. Our aim is to demonstrate that EM brings far greater conceptual unity to the interactions of the student, the teacher and the developer than is typically found in using a conventional approach to educational software development.

1. Introduction

The ideal of constructionist computer-assisted learning [3] can be seen as unifying three roles: that of the student, the teacher and the developer. The learner first explores in ignorance and confusion in the role of a student, then identifies concepts and objectives for model-building to support and direct their exploration, then constructs appropriate models with which to repeat a similar cycle of interaction (see Figure 1a). Without a suitable end-user programming environment to meet these ideal demands, many educational studies have sought to realise some of the benefits of the constructionist vision by bringing together a developer and a teacher (or 'teacher-developer') working with students to implement this development cycle. This approach encounters several problematic issues:

- collaboration cannot realise the benefits of one individual *being* at one and the same time student, teacher and developer;
- the activities being carried out by student, teacher and developer typically have a widely divergent nature – corresponding to the use, specification and implementation of a program respectively;
- in a conventional programming framework, the development cycle is notoriously difficult to manage effectively, and the well-recognised problems of adapting programs to new requirements are particularly challenging in the educational setting, where subtle changes to the requirement are both commonplace and of the essence.

The merits of using Empirical Modelling (EM) principles in the development of educational technology have been discussed at length in several previous papers [1,2]. A key idea is that EM establishes a much closer connection between the model-building and the learning, so that the same kinds of insight that assist the student and teacher also aid the developer in model-building. It would be misleading to suggest that this convergence of roles is yet meeting the full objectives of the constructionist agenda, but it can be seen as a significant step in this direction. In many contexts, there is a clear sense in which the interactions of student, teacher and developer take essentially the same abstract computational form and differ only in how they are to be interpreted (cf. Figure 1b). The primary purpose of this paper is to demonstrate how this principle operates explicitly in a specific learning context, namely, understanding an algorithm that is used in relational database design.

1.1. The Testing Lossless Join Algorithm

In designing a relational database, an important consideration is whether a particular decomposition of a relation R into subschemes R_1, R_2, ... , R_k has *lossless join*. This property holds if, for any given extension r of R satisfying all the functional dependencies (FDs) that hold in R, the natural join of the projections of r onto each of the k sub-schemes is r itself. As a simple

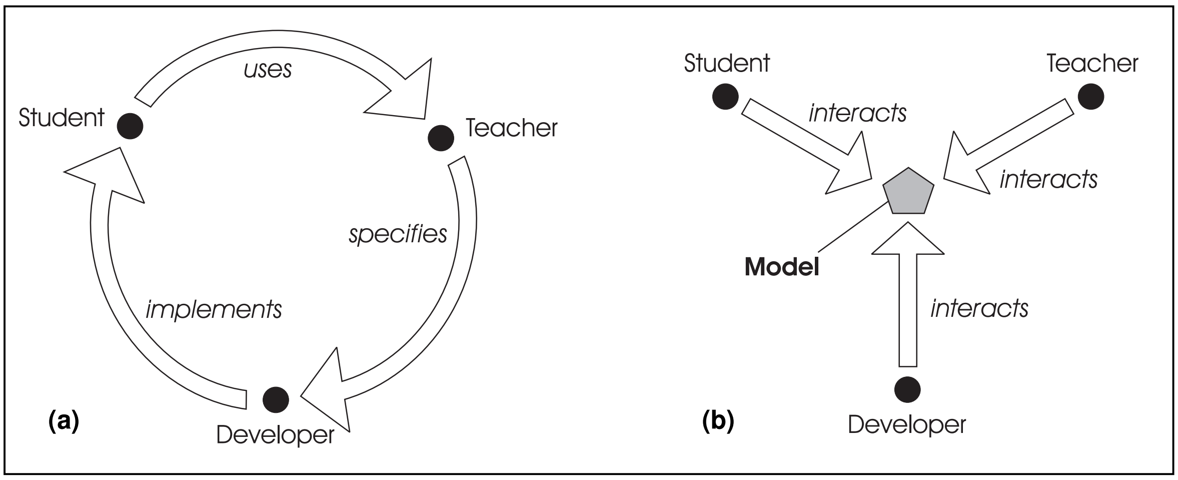


Figure 1: Roles in constructionist learning: (a) traditionally; (b) in model construction using EM

illustration, consider the relation R with attributes Supplier, City and Agent, subject to the FDs S→C and C→A. The decomposition SC, CA of R is lossless, since if (s_1,c_1) and (c_1,a_1) are derived from R by projection onto SC and CA, then (s_1,c_1) must be derived from a tuple (s_1,c_1,a_2), and (c_1, a_1) from a tuple (s_2,c_1,a_1). Because the tuples (s_1,c_1,a_2) and (s_2,c_1,a_1) agree on the attribute C, the FD C→A ensures that $a_2=a_1$, so that the natural join of SC and CA is necessarily R. In contrast, the decomposition SA, CA of SCA is *lossy* ('non-lossless'): to prove this it is only necessary to exhibit an agent *a* who is associated with the city *c* at which supplier *s* is based, and who also serves as the agent for a second city *c'* at which the supplier *s'* is based. The three tuples (s,a), (c,a) and (c',a) then feature in the projections of R onto SA and CA, so that the 'rogue' tuple (s,c',a), inconsistent with the FDs on R, appears in the natural join of SA and CA.

The Testing_Lossless_Joins (TLJ) algorithm, as specified in Ullman [5] (see Algorithm 7.2 on p227), is a standard component of the relational database theory. The essential principles of the algorithm can be inferred from the following brief informal description. The first stage of the algorithm is to set up an array in which each entry is a symbolic element of the form a_i or b_{ij}, where i (respectively j) is the index of the row (respectively column) in which the element is located, and an a_j appears in location (i,j) if and only if the attribute associated with the j-th column appears in the i-th subscheme. The algorithm then proceeds step-by-step by taking account of the FDs in turn in cyclic order. At each step, when a particular FD of the form X→Y is under consideration, the array is processed in such a way that if any two rows have identical entries in the columns associated with all the attributes in X, they are modified so that they also agree on all attributes in Y. In this process of modification, b_{ij} entries are replaced by a_j entries wherever possible, and agreement is otherwise established by assigning the same indices to all the relevant b_{ij} entries. The algorithm terminates when no further modification of the array results from the application of any of the given FDs, at which point the join is declared lossless if and only if there is a row comprised of a_j entries.

For the purposes of this exposition, without loss of generality, all FDs will be assumed to be of the form X→S, where S is a single attribute. The representation of the entries in the table can also be simplified so that they have numeric values. Specifically, a's and b's can be represented by integers: each a by 1 and the initial b entries by integers greater than 1. When several values are to be equated, it is then appropriate to equate all values to the least. This representation, which is suited to computer implementation, is valid since the indices of a's and b's are redundant, and all comparisons are made between elements in the same column.

2. Computer support for TLJ

The TLJ algorithm is in some respects a natural target for computer support. For instance, a student who is exercising the algorithm (or a teacher who is demonstrating the algorithm) typically indicates the successive modifications that are made to the array by crossing out entries and inserting their new values until such time as the array entries become difficult to read, then making a new copy of the array and repeating the annotation process. This is an error-prone process that does not always give a clear indication of the precise steps carried out. It is easy to see that, when we consider the possible motivations and issues that arise in learning, presenting or assessing the TLJ algorithm, the list of requirements becomes very large. The teacher alone will typically want: a dynamic way of presenting the algorithm that draws attention to the specific observations and actions that are being carried out at each step; to be able to simulate the operation of the algorithm in full; to be able to experiment with different sets of FDs, perhaps even whilst the algorithm is being executed; to emulate errors that a student might make in exercising the algorithm; to use the model as the basis for exercises that test a student's understanding as comprehensively as possible. In devising exercises or an examination question, the teacher will not wish to restart the algorithm from scratch at each new iteration required in the design. And unless they enjoy the dedicated support of a developer, they will ideally want to be able to adapt the model relatively painlessly themselves to take account of different perceptions of what the student requires, and of any special, possibly idiosyncratic, misunderstandings they have.

As a requirement for a conventional program, this presents a formidable challenge. What is more, it is quite apparent that the requirement is not in any sense complete. In constructing a conventional program to meet these needs, there will invariably be optimization for specific purposes that will prove obstructive to future extensions. The key to addressing this problem is to recognize that what is required of a model to support the learning of the TLJ algorithm is a form of automation that can integrate fully with the activities that the teacher can perform manually as the need arises. In effect, this allows human discretion and intelligence to be exercised in situations where there is no satisfactory preconceived fully automated solution that can be applied. This is the function of the EM *construal* for the TLJ algorithm to be described below.

2.1. An EM construal for the TLJ algorithm

An EM *construal* is a computer-based model that embodies the patterns of observation, dependency and agency that are observed in its referent [1]. A detailed account of the principles and tools used in developing construals in EM is beyond the scope of this paper (see [6] for more details), but the essential ideas can be illustrated with reference to our chosen case-study.

For the TLJ algorithm, the primary observables are the contents and attributes of the table that is generated in executing the algorithm and the FDs that are associated with these attributes. Both teacher and student come to understand the algorithm in terms of just these observables; building a construal to embody these observables, and the patterns of dependency and agency to which they are subject, is also a most appropriate way for the developer to provide support for the manual, semi-automated or fully automated interaction that must accompany the learning of the algorithm.

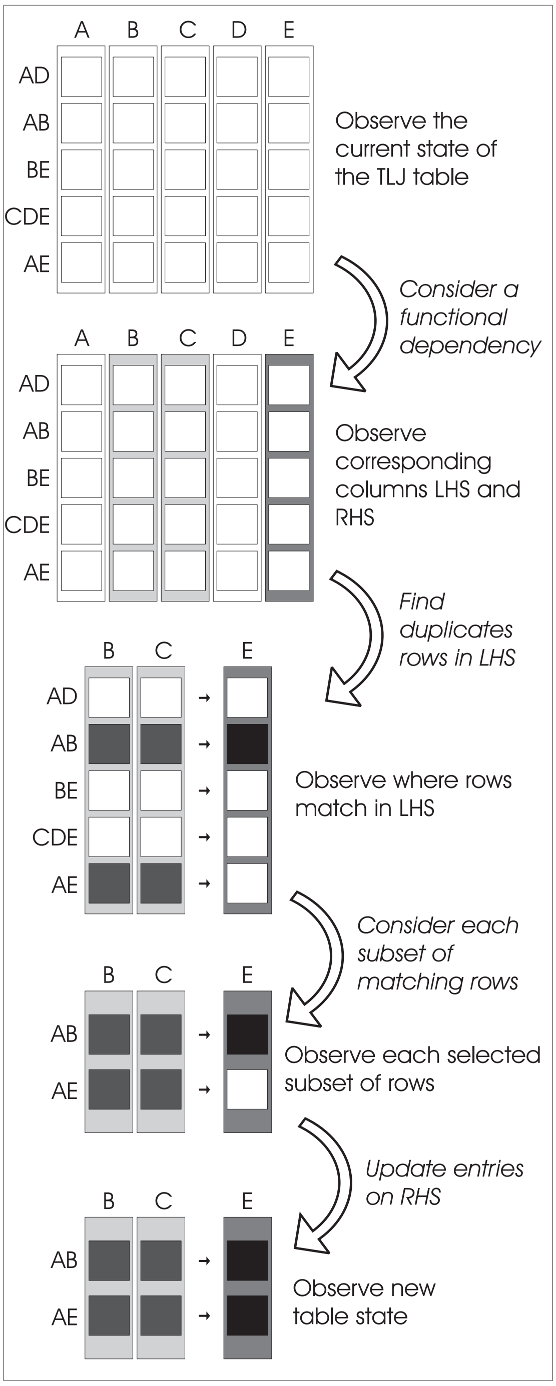

A B C D E
AD
AB
BE
CDE
AE
Observe the current state of the TLJ table
Consider a functional dependency
Observe corresponding columns LHS and RHS
Find duplicates rows in LHS
B C E
Observe where rows match in LHS
Consider each subset of matching rows
Observe each selected subset of rows
Update entries on RHS
Observe new table state

Figure 2: The TLJ pattern of observation

```
project_table_LHS_FD is project(current_table, makestrlist(FDs[current_FD][1]));
project_table_RHS_FD is project(current_table, [FDs[current_FD][2]]);
pattern_duplicate_rows is index_duplicated(tail(project_table_LHS_FD));
newcol is transformcol(makelistcol(project_table_RHS_FD), pattern_duplicate_rows);
newtable is apply_current_FD_current_table(current_table, newcol);
```

Listing 1: Observables and dependencies in the TLJ construal

Learning the TLJ algorithm is linked to a pattern of observation that applies at each iteration. The learner consults the current state of the table with a specific FD X→S in mind, observes the pattern of tuples that arises in the columns associated with the left-hand side X of the FD to detect where there are duplicates, then observes how this pattern applies to the column associated with the right-hand side S of the FD. The core step of the algorithm is the substitution of the resulting transformation of the column associated with S for the original column.

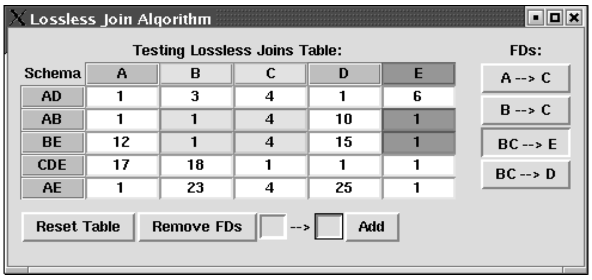
For a particular table and FD, the above ingredients of the core pattern of observation can be displayed pictorially as in Figure 2. The arrows in this figure represent dependencies between observables, expressing the way that a given state of the TLJ table, and a given FD determines the set of columns LHS and a column RHS, and how the duplicate rows in the set of columns LHS then determine the updated entries in the column RHS. In the modelling environment used to develop the construal, these dependencies can be directly specified and are automatically maintained (cf. the spreadsheet principle). This makes it possible to explore, in an experimental fashion, the way in which the current instance of this pattern of observation is affected by changing the current state of the TLJ table, or the current FD.

3. Developing and deploying the construal

The exploratory activity that surrounds the identification of observables and dependencies is a core activity that is central to the interests of the student, the teacher and the developer. As Figure 2 illustrates, the contexts for observation with which the student must become familiar in learning the TLJ algorithm are rich and subtle: they involve moving from global observation of the entire table to localised observation of the entries in specific rows and columns. It is also significant that the activities denoted by the arrows in Figure 2 are best conceived as mental operations on the part of the student, preparatory to the action of updating the table. From a teacher's perspective, each of the arrows can be interpreted as a link in a chain of observation involved in executing a step of the TLJ algorithm. As such, it can be the subject of an exercise: for instance, identifying the columns LHS and RHS, given a table and a FD. Decomposing the pattern of observation into a chain of simpler observations also has potential value as a diagnostic tool: for instance, helping the teacher to detect where a student understands the updating mechanism correctly, but is mistaken in their interpretation of a FD relation.

From the perspective of this paper, the relevance of Figure 2 for the developer has particular interest. There is a very direct correspondence between Figure 2 and the EM construal for the TLJ that was first constructed as an open interactive environment by the first author, and subsequently extended by the second to provide specific interfaces to the construal such as that depicted in Figure 3. This correspondence is best appreciated by interacting with the dynamic script development environment that is supported by the EM tool used in this development: the `tkeden` interpreter [6], but it is to some extent apparent from the relationship between Figure 2 and Listing 1. Just as the pattern of observation depicted in Figure 2 is the core of the TLJ algorithm, so the script of five definitions linking observables and dependencies in Listing 1 is the core of the TLJ construal. The names of the observables in Listing 1 have been made more expressive, and the code for operators (such as `index_duplicated`, and `makelistcol`) has been omitted, but the definitions are essentially as they appear in the `tkeden` source. Since our aim is to illustrate the convergence of viewpoints of student, teacher and developer suggested by Figure 1b, a brief explanation of how this script was developed, and relates to the pattern of observation in Figure 2, is appropriate.

As is evident by inspection, the values of all the observables in the script in Listing 1 are determined from the index of the FD that is currently of interest (`current_FD`) and the current contents of the TLJ table (`current_table`). The first two definitions determine the contents of the columns that correspond to the LHS and RHS of the current FD respectively. The third definition identifies the pattern of duplicate rows in the columns in the LHS of the FD; the fourth


Testing Lossless Joins Table:

Schema	A	B	C	D	E
AD	1	3	4	1	6
AB	1	1	4	10	1
BE	12	1	4	15	1
CDE	17	18	1	1	1
AE	1	23	4	25	1

Figure 3: Screenshot of the TLJ model

expresses the way in which the new contents of the RHS column is to be updated by consulting the pattern of duplicate rows. The final definition expresses the relationship between the original value of the table and the value that it takes after the FD has been processed. These definitions correspond closely to the links in the pattern of observation in Figure 2: in establishing the definitions using the `tkeden` interpreter, the operators introduced to specify the relationship associated with each link are tested in isolation by supplying different test values for the parameters in much the same way that the student might confirm that they have understood each observational link in mastering the algorithm. Though the development otherwise has more of the characteristic flavour of conventional programming, it remains anchored in this way to the learning domain. The missing elements of the `tkeden` source are the specifications of the operators themselves, which take the form of rather straightforward procedural code to compute an output from an input without side-effect. The script illustrates other features that are of interest from a computational perspective. These include:

- the re-use and adaptation of standard operators (such as the relational operator `project`, borrowed from the relational database extension of `tkeden`).
- the use of definitions to maintain dependencies between different modes of observation that are a common concern for traditional programmers, namely those that are associated with two or more data structures for a particular application (such as the conversion function `makelistcol`).

For the experienced developer using `tkeden`, the model-building task is greatly simplified by a combination of these three techniques: programming of relatively simple functions without side-effects; re-use of existing functions and scripts; and the use of definitions to maintain many different consistent concurrent representations of a given family of observables.

4. Conclusion

The difficulty of unifying the roles of student, teacher and developer is one of the obstacles to constructionist computer-assisted learning. The technical problems of supporting the degree of openness in interaction that constructionism ideally presumes are so acute that Ehrmann [4] has been led to question whether the vision of learners constructing their own learning environments is a mirage. It is clear that in activities such as developing micro-worlds for children – at any rate with current software tools – there is little prospect that the learners can themselves carry out the model construction. Our case study is of interest because it proves that in principle there can be a high degree of synergy between interactions that are demanded of the learner in the roles of student, teacher and developer. For the target group of learners (viz. undergraduates with high levels of programming skill following an advanced module in database theory), there is no great conceptual or practically significant distinction between the kind of activity involved in learning about the lossless join algorithm and that involved in constructing the associated EM construal. It remains to be seen to what extent, subject to appropriate tool refinement and suitable training in the application of EM principles and tools, the same synergy between learning and model-building can be demonstrated in other learning contexts.

5. References

[1] W.M.Beynon, C.Roe, Computer Support for Constructionism in Context, In *Proceedings of the 4th IEEE International Conference on Advanced Learning Technologies* (ICALT), 2004, 216-220.

[2] W.M.Beynon, Empirical Modelling for Educational Technology, In *Proceedings of Cognitive Technology*, University of Aizu, Japan, IEEE, 1997, 54-68.

[3] B.Dalgarno, Constructionist Computer Assisted Learning: Theory and Techniques, In *Proceedings of the Australian Society for Computers in Learning In Tertiary Education*, 1996.

[4] S.Ehrmann, Technology & Revolution in Education: Ending the Cycle of Failure. *Liberal Education*, Fall, 40-49.

[5] J.D.Ullman, *Principles of Database Systems*, Computer Science Press, 1982.

[6] `www.dcs.warwick.ac.uk/modelling`