Pre-publication draft of a paper which appeared in The Proceedings of the 8th Annual Conference of the Australian Society for Computers in Learning in Tertiary Education (ASCILITE 91)

# INSIDE THE UNIFICATION TUTOR:
# THE ARCHITECTURE OF AN INTELLIGENT EDUCATIONAL SYSTEM

**Geoffrey I. Webb**
*Department of Computing and Mathematics, Deakin University, Geelong 3217.*

## Abstract

The Unification Tutor provides practice and tuition on the unification of terms from the Prolog programming language. It integrates multiple knowledge sources encompassing both performance and declarative knowledge. A key feature of the tutor is the use of a detailed student model. It has been used since 1989 in Computer Science courses at Deakin and La Trobe Universities. Previous papers have examined the student modelling component of this system. This paper investigates the internal operation of the Unification Tutor, the sub-systems it incorporates and their interaction.

**Keywords**
Artificial Intelligence, Student Modelling, Intelligent Tutoring Systems

## 1 Introduction

The Unification Tutor is an experimental educational system that incorporates a number of artificial intelligence based components. The primary objective in developing the system has been to refine and evaluate the feature based modelling (FBM) student modelling technique (Webb, 1989, 1991). In consequence, one of the key components of the system is a FBM modelling system. Other components include a knowledge base; two knowledge-based feedback generation systems and a task generator.

The Unification Tutor examines the unification of terms from the Prolog programming language. Unification is the key process that drives Prolog. It is a relatively simple yet non-trivial problem solving skill.

Previous papers have described student use of the system (Webb, Cumming, Richards & Yum. 1989, 1990). This paper describes the internal operation of the system. It is ordered as follows. The next section describes Feature Networks, the knowledge representation formalism used in the tutor. This is followed by a description of FBM. A brief description of the unification of Prolog term then sets the scene for a description of the architecture of the system and examinations of the individual components and how they interact.

## 2 Feature Networks

Feature networks are a simple knowledge representation formalism that describe the important features of a domain and how they inter-relate (Webb, 1988). Features may be thought of as properties or attributes of the elements of the domain.

Features are grouped by a feature network through the use of feature choices. A feature choice represents a set of epistemologically related but disjoint features. For example, a feature choice in a domain relating to the repair of electronic equipment might indicate whether the power is turned on by grouping the two features *the power is on* and *the power is off*.

The other elements of a feature network describe the relationships between features and feature choices. They may be considered in terms of specifying specialisation and generalisation relationships.

A set of features $f$ a specialisation of a set of features $g$ if and only if all the features in $g$ apply to every possible object to which all the features in $f$ apply and all the features in $g$ also apply to objects to which all the features in $f$ not apply. For example, the feature *the widget light is on* would be a specialisation of *the power is on* and *the widget switch is on* if it were not possible for the widget light to be on unless the power and widget switch were on and it was possible for the power and widget switch to be on but the widget light to be off.

If a set of features $f$ is a specialisation of a set of features $g$ then $g$ is a generalisation of $f$. Note that the terms *specialisation* and *generalisation* are used here to mean proper specialisation and proper generalisation thus excluding the possibility that for any two sets of features $f$ and $g$, $f$ is both a specialisation and generalisation of $g$.

Two types of features are distinguished - task features and action features. Task features describe relevant properties of tasks that a student may undertake. Action features describe relevant properties of actions that a student may perform while engaged in those tasks.

## 3 Feature Based Modelling

Whereas most previous approaches to cognitive modelling (for example, Brown & Burton, 1978; Clancey, 1987; Goldstein, 1979; Reiser, Anderson & Farrell 1985; Sleeman, 1984; Stevens, Collins & Goldin, 1982; VanLehn, 1982) have sought to model the internal operation of the cognitive system, FBM seeks to model the cognitive system at the level of input and output by creating a model of the relationships between the tasks on which a student engages and the actions that they perform while engaged in those tasks.

This model is represented by a set of associations. Each association ($T \longrightarrow a$) relates a set of task features $T$ to an action feature $a$ and indicates that when all of the features in $T$ apply to a task, $a$ applies to the student's actions while engaged in that task. For example, *{the power is off the widget switch is off}*$\longrightarrow$*toggle the widget switch* indicates that when the power and widget switch are off the student toggles the widget switch.

The cover of an. association $T \longrightarrow a$ is the set of all tasks to which $T$ applied and for which $a$ applied to the student's actions. The cover of a set of task features $T$ is the set of all tasks to which $T$ applied.

Two types of association are distinguished - appropriate associations and erroneous associations. Appropriate associations are associations that the student is desired to have, that is where it is desired that the student always act in the prescribed manner when the prescribed task features are present. All other associations are erroneous.

The student model contains all associations. However, key associations can be identified which usefully summarise the model. An association $T \longrightarrow a$ is a key association if and only if –

- for every association $U \longrightarrow a$ such that $U$ is a generalisation of $T$, the covers of $T \longrightarrow a$ and $U \longrightarrow a$ are identical;
- for every association $V \longrightarrow a$, such that $V$ is a specialisation of $T$, the cover of $V \longrightarrow a$ is a subset of the cover of $T \longrightarrow a$; and
- for every association $T \longrightarrow b$, such that $b \neq a$, $b$ is a generalisation of $a$.

To allow for noise (small numbers of interactions that are not representative of the student's underlying mastery of a domain, for example, due to slips on a keyboard) an association is allowed in the presence of some counter-evidence, so long as there is no regularity detected in that counter-evidence (see Webb, 1991, for further details).

## 4 The Unification of Prolog Terms

Unification is a key process behind the operation of the Prolog programming language. There are two types of Prolog terms, variables and compound terms. A variable is represented by a sequence of alpha-numeric characters starting with an uppercase alphabetic character. A compound term consists of a functor and a sequence of arguments. A functor has a name, which is represented by a sequence of alpha-numeric characters starting with a lowercase alphabetic character, and an arity, which indicates the number of arguments that it requires. Atoms are compound terms with arity 1, for example `anAtom`. `Avariable` is an example of a variable and `a(Compound, term, with ( arity, Three))` is an example of a compound term whose three arguments are a variable, an atom and a compound term, respectively.

A substitution is a set of substitution pairs, each of which relates a variable *V* with a term *T*, written *V*=*T*, A variable that appears on the le ft of a substitution pair cannot appear elsewhere in the substitution.

To apply a substitution *S* to a term *T* requires that every variable in *T* that appears on the left of a substitution pair in *S* be replaced by the right-hand-side of that pair. For example, `an(Example, Term) {E = X, Term = result} = an (Example, result)`.

A unifier of two terms is a substitution that, when applied to either of the two terms, provides the same answer. A most general unifier for two terms is a unifier for those terms that contains the minimum possible number of substitution pairs.

Many most general unification problems have multiple correct solutions. For example, **{X=Y}** and **{Y=X}** are both most general unifiers for **X** and Y.

## 5 System Architecture

The Unification Tutor operates by repeating the following sequence of operations until either the student stops the lesson or the student model indicates that the student fully understands all aspects of unification. First, a problem is selected and presented to the student. Each problem consists of two Prolog terms to be unified. Next the student interacts with the student interface to provide an answer. This answer is analysed and the student model is updated. Feedback is then presented to the student based on analyses of the problem, the student's answer, and the student model.

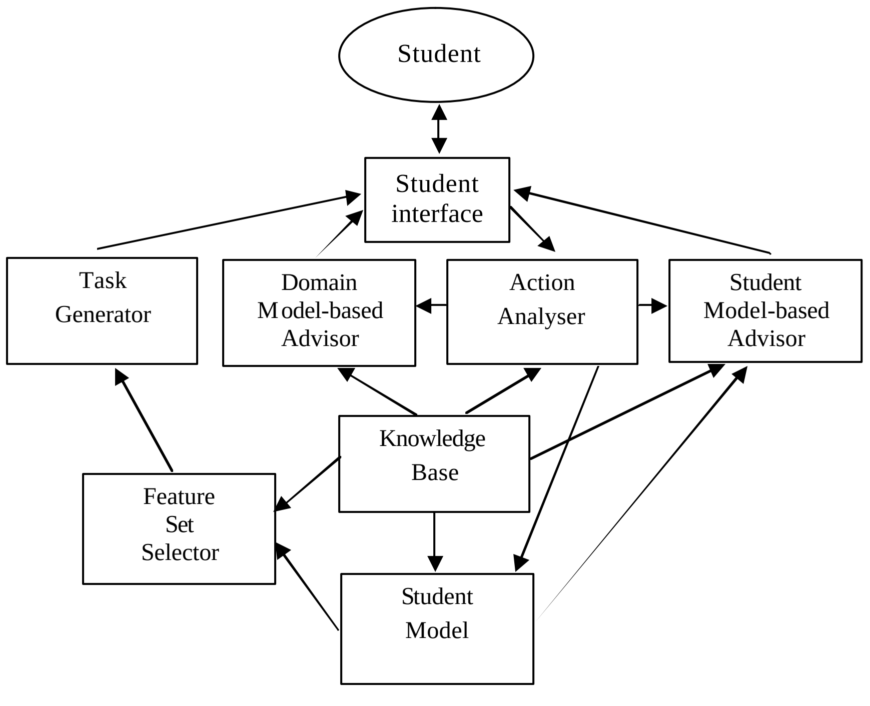

**Figure 1: Major components and information flow.**

Figure 1 presents the major components of the tutor and the flow of information between them.

The knowledge base consists of a feature network describing the domain of unification augmented by additional specialisation and generalisation information. This feature network contains 37 task features and 16 action features.

The feature set selector examines the current state of the student model and selects a set of features, the examination of which will best advance the student's understanding of the domain. In doing so it seeks to avoid tasks that the student can solve without difficulty as well as tasks whose solution is too complex for the student to reasonably tackle. To achieve the first of these ends, it avoids the presentation of problems whose task features are associated in the student model with the action feature *correct* (indicating that the student's actions while tackling such problems are appropriate). To avoid problems that are too complex, the system does not present problems whose solution requires the solution of sub-problems whose features are not associated with *correct*. In consequence, any difficulty that the student experiences should always arise from the global features of the current problem and how it combines sub-problems, rather than from difficulties with any of the sub-problems in isolation. This latter measure also assists the modelling system, as it enables the details of sub-problems to be ignored while constructing the student model in favour of the features of the global problem and the inter-relationships between the immediate sub-problems.

Early versions of the feature set selector attempted to select the least difficult problems that were covered by the task features of an erroneous association, on the assumption that examining problems covered by the task features of an erroneous association would help the student correct that association and that the less difficult such a problem the easier it would be for the student to analyse and utilise.

However, it became apparent that for many erroneous associations $T \rightarrow a$, *a* would be appropriate for the least difficult tasks covered by *T*. For example, the erroneous association *{the_terms_contain_variables}*$\rightarrow$*unify_the_terms* states that when two terms contain a variable the student provides a unifier. However, the least difficult problems of this type are cases in which one of the terms is a simple variable in which case the association leads to a correct answer. Thus, if the old version of the system detected this association it would tell the student that it was not appropriate to always provide a unifier for problems that involved a variable and then present a series of such problems in which it was appropriate to do exactly that. If anything, the system tended to reinforce and encourage the erroneous associations that it detected.

In response to this deficiency the feature set generator was altered so as to prefer problems that were covered by the task features of an erroneous association and for which the action feature was not appropriate.

The student interface manages all interactions with the student. It includes a help subsystem and facilities for editing unification problem solutions. It accepts a problem from the task generator, presents it to the student, accepts a response from the student and passes it along with the problem description to the action analyser. The student interface contains a parser that determines whether the student's input is valid and, if so, transforms it into an internal representation for use by the rest of the system.

The action analyser examines the student's solution to derive a set of action features that describe it. A number of methods are used during this analysis. The action analyser includes its own local knowledge in the form of two subsystems that can solve different types of problems relating to the domain. The unification solver is passed the two Prolog terms that represent a task and returns a most general unifier for those terms. The substitution solver is passed a Prolog term and a substitution and returns the result of applying that substitution to the term.

To determine whether a student's solution is a unifier, the action analyser applies the student's answer to both terms from the problem to determine whether the results are identical. Only if they are, has the student provided a unifier.

To determine whether a unifier is most general, the action analyser compares the student's answer to the answer provided by the unification solver. If the student's answer contains more substitution pairs than the unification solver's answer, the solution is not most general.

The information gained from these tests and a number of syntactic analyses of the student's solution allow the action analyser to determine which of 16 action features describes a student's solution. Examples of the action features used include *the_answer_is_{},the_answer_is_overspecialised*, and *the_answer_is_correct*. Note that the first of these is sometimes appropriate and sometimes inappropriate, the second is never appropriate and the last is always appropriate.

Once an answer has been analysed, the task features, appropriate action features and observed action features are passed to the student modelling sub-system that uses this information to update its FBM model of the student.

Next the domain model based advisor is invoked. This sub-system examines the student model to determine whether there is a suitable association to discuss with the student. An association $T \rightarrow a$ is considered suitable for discussion if -

- it is erroneous;
- it covers the most recent interaction;
- $a$ was not appropriate for that interaction; and
- it is a key association (see the Feature Based Modelling section, above).

If such an association is found it is discussed with the student. Such discussion currently consists of describing the association to the student, stating that this is not appropriate as demonstrated by the most recent problem, describing how it is not appropriate, describing the systems solution and suggesting that the student reconsider his approach to such problems. Figure 2 illustrates such an interaction. The student's response is underlined.

```
Enter a most general unifier for the following terms or type none, ? or exit.
Third
i
⇒ none
It appears to me that when two terms are different you state that the terms do
not unify.
This is not always appropriate as you can see from the above example for which
you should provide a unifier.
My answer is {Third=i}.
Press space to continue.
```

**Figure 2: Student model based feedback.**

If the student model based advisor is unable to find a suitable association to discuss with the student, the domain model based advisor presents advice instead. The domain model based advisor presents advice to the student based solely on the details of the current task, the students solution and the system's solution. If the student's solution is correct (which is determined by examining the *Correctness* feature choice), one of a number of stored positive messages is randomly selected and presented to the student. If the solution is incorrect, one of the action features that differs from the appropriate action features for the task is chosen and the error that the difference represents is described to the student. The action features are ordered by the lesson author in terms of their likely importance to enable the system to select one from many. Following the description of the error, the system's solution to the problem is presented. Figure 3 presents such an interaction. The student's response is underlined.

In all, the system contains three distinct domain knowledge sources. The feature network is a description of the properties of tasks in the domain and the manner in which they interact. The unification solver embodies procedural knowledge for solving problems in the domain. Finally, the substitution solver embodies procedural knowledge for solving substitution problems, knowledge that is helpful in analysing the student's responses.

```
Enter a most general unifier f or the following terms or type none, ? or
exit.

list(C, a(atom, C), sub_list(Result))

list(sub_list(result), a(atom, TheValue), Thevalue)

⇒{C=sub_list(Result),TheValue—sub_list(Result), Result=sub_list(Result)}

A substitution should never contain the same variable on both the left and the
right of a substitution pair. Your answer has Result in both these positions.

My answer is {TheValue=sub_list(Result), C=sub_list(Result)}

Press space to continue.
```

**Figure 3: Domain model based feedback.**

## 6 Conclusion

The Unification Tutor utilises multiple knowledge sources to provide tuition in the unification of Prolog terms. The student model is used both for the selection of suitable tasks for the student to tackle and for generating advice for the student. Two procedural knowledge sources are used for analysing the student's answers and generating correct solutions to the tasks posed.

The system has been used by one class of third year Computer Science students at La Trobe University and three classes of third year Computer Science students at Deakin University. Student response to the system has been favourable (Webb, Cumming, Richards & Yum, 1990).

The development of this system with the limited resources at our disposal demonstrates that the application of artificial intelligence techniques to education is both feasible and practical.

### Acknowledgements

This research has been supported by grants from the Australian Research Council. The research has been conducted as part of the EXCALIBUR artificial intelligence in education research project and has benefited from comments and advice from my EXCALIBUR co-researchers Tom Richards, Geoff Cumming and Roly Sussex.

### References

[1] Brown, J.S. & Burton, R. R. (1978) **Diagnostic models for procedural bugs in basic mathematical skills**. *Cognitive Science*, Vol. 2, pp. 155-192.

[2] Clancey, W.J. (1987) *Knowledge-Based Tutoring: The GUIDON Program*. MIT

[3] Press, Cambridge, Mass.

[4] Goldstein, I. P. **The genetic-graph: A representation for the evolution of procedural knowledge**. *International J. Man-Machine Studies*, Vol 11(1979), pp. 51-77.

[5] Reiser, B. J., Anderson, J. R. & Farrell, R. G. (1985) **Dynamic student modelling in an intelligent tutor for Lisp programming**. *Proceedings of the Ninth international Joint Conference on Artificial Intelligence*, pp. 8-14.

[6] Sleeman, D. H. (1984) **Inferring student models for intelligent computer-aided instruction**, in R. S. Michalski, J. G. Carbonell & T. M. Mitchell (Eds.) *Machine Learning: An Artificial Intelligence Approach*, Springer-Verlag, Berlin, pp. 483-510.

[7] Stevens, A. L., Collins, A. & Goldin, S. E. (1982) **Misconceptions in students' understanding.** In D. H. Sleeman & J. S. Brown (Eds.) *Intelligent Tutoring Systems*, Academic Press, London, pp. 13-24.

[8] VanLehn, K. **Bugs are not enough: Empirical studies of bugs, impasses, and repairs in procedural skills**. *Journal of Mathematical Behaviour* Vol. 3 (1982), pp. 3-72.

[9] Webb, G I. (1988) **A knowledge-based approach to Computer-Aided Learning**. *International Journal of Man-Machine Studies*, 29: 257-285.

[10] Webb, G I. (1989) **Cognitive diagnosis using student attributions**. *Proceedings of the Second Australian Society for Computers in Learning in Tertiary Education Conference*, Canberra. pp. 502-514.

[11] Webb, G I. (1991.) **An attribute-value machine learning approach to student modelling**. *Proceedings of the IJCAI Workshop W.4: Agent Modelling for intelligent Interaction*, Sydney. pp. 128-136.

[12] Webb, G I., Cumming, G., Richards, T. J. & Yum, K-K. (1989). **The Unification Tutor - An intelligent educational system in the classroom**. *Proceedings of the Seventh Annual Conference of the Aust. Society for Computers in Learning in Tertiary Education*, Gold Coast, pp. 408-420.

[13] Webb, G I.., Cumming, G, Richards, T. J. & Yum, K-K. (1990). **Educational evaluation of Feature-Based Modelling in a problem solving domain**. To be published in *Proceedings of the Conference on Advanced Research on Computers in Education*, Tokyo.