In U. Schmid, J. Krems, & F. Wysotzki (Eds.), *Proceedings of the First European Workshop on Cognitive Modeling*. 22-30. Berlin: Forschungsberichte des Fachbereichs Informatik, Technische Universität Berlin.

Able III: Learning in a more visible, principled, and reusable way

Frank E. Ritter (ritter@psychology.nottingham.ac.uk)
Gordon D. Baxter (gdb@psychology.nottingham.ac.uk)
++44 (115) 951 5302; Fax: 951 5324
ESRC Centre for Research in Development, Instruction and Training
Department of Psychology, University of Nottingham, Nottingham NG7 2RD, UK

Abstract

We describe an improved version of the Able cognitive model that exhibits a novice to expert transition in solving physics problems. The initial model was written by Larkin and initially translated into Soar (ver. 4) by Levy. In revising it to run in the latest version of Soar (7.0.4), we have updated it to be an exemplar of an understandable and reusable cognitive model. It includes graphic displays for indicating how it works. Able's learning mechanism has been organized into a general learning utility for use in other models where further principles can be specified (we demonstrate this for a new problem). Finally, it is publically available. We argue that this example standard of displays and reusability must be realized more often if cognitive modeling is to prosper.

Introduction

There are serious problems restricting the use of cognitive models. It is probably fair to say that most cognitive models are not reused, even when they are created in a cognitive architecture designed to facilitate their reuse. It is also probably fair to say that cognitive models can often be difficult to explain and understand. There are, of course, numerous exceptions[1], but overall, cognitive modeling does not have the system reuse that the AI community achieves.

We describe here a revision of a previously presented cognitive model. In this revision, we do not expand the model to cover more data directly, but to be more useful. We do this by (a) making the model easier to understand; and (b) making the model easier to apply to additional data, that is, reuse it. We demonstrate its reuse here on a small sample task, but we had used before attempting to make it into a utility. We believe that including these two features will set a new standard for models, and further fulfills the promise this model initially offered, that of providing an account of knowledge application and compilation in formalizable domains.

We are using a model of physics problem solving called Able (Larkin, 1981; Larkin, McDermott, Simon, & Simon, 1980b). Able solves kinematics problems by applying physics principles. It initially uses a backward-chaining, means-ends analysis to find which principles to apply, starting with the target variable; after learning, it ends up with a more expert-like behavior, solving problems without search. While Able does not model the complete process, such as learning the principles, setting up the problem, and performing the algebraic manipulations, it is does model fairly well on a high level the novice to expert transition of principle application in formalizable domains like physics problem solving.

Able was initially written as two related models: ME to simulate novice physics problem solvers (barely able); and KD to simulate expert problem solvers (more able) in kinematics (Larkin, et al., 1980b) and fluid statics (Larkin & Simon, 1981). These models were compared with problem solving protocols, which they matched very well. The models were later unified by a learning mechanism that learned while solving problems, showing how the novice model could become an expert model through practice (Larkin, 1981). This unified Able model was translated into Soar 4 by Levy (1991), showing that the learning mechanism used by Larkin was essentially the chunking mechanism in Soar (Newell, 1990). Levy's work remains an interesting example of how quickly someone can learn and model in Soar, for he wrote it in two weeks. His model is where we started.

What we want from a cognitive model

We have previously reused the principle application mechanism in Able to include a simple reasoning component in a model that solved a simple air traffic control-like task (Bass, Baxter, & Ritter, 1995). Because we found Able's mechanism so useful and because we will be creating further models that will have multiple places to apply this reasoning mechanism, we felt it was worthwhile to regularize Able and create a utility for our own use.

There are several things that we want from such a modeling utility. We want it to match or explain some data and make predictions about additional behavior. Able already does this. We also want to include learning, particularly learning that could implement the transition from novice

[1]The Symbolic Concept Acquisition version 2 model by Pearson (based on Miller's work) and the Subtraction model by Jones (based on Brown, VanLehn's, Young and O'Shea's work) from the U. of Michigan are nice exceptions that also inspired this work. They include explanatory displays and their code is available. The models (or their authors) are available through http://ai.eecs.umich.edu/soar/soar-group.html. The PDP toolkits that provide displays are exceptions as well.

(backward-chaining) behavior, to expert (forward-chaining) behavior (e.g., Klein, 1989). Able already does this as well.

We also want the model to be easier to understand and to explain. In order to do this, we created some graphic displays in the scripting language associated with Soar. Some of these displays are specific to Able, and some of the displays can be used with other models. When we first reused Able (Bass, et al., 1995), doing so was not difficult, but it was not as straightforward as we might have liked. The bulk of our effort reported here went into regularizing Able's structure so that it could provide a general mechanism that could be routinely reused. We demonstrate below how we were able to use it very quickly to model a simple task in a similar domain.

Lastly, one thing you might want from a cognitive model is the model itself. If a unified theory of cognition is going to provide an approach that supports integrating models (Newell, 1990), the models must be available. The clearest way to support this is to put the documented model in a public place. Our version of Able, Able III, with its associated displays, principle application utility, and a small additional example model is available from ftp://www.ccc.nottingham.ac.uk/pub/soar/nottingham/. It is useful as an example Soar model, and as a utility for creating additional models by adding principles. The general graphic displays will be useful to anyone creating, examining, or teaching Soar models. It currently runs with the latest Soar 7.0.4 release under Unix and MacOS.

How Able works

Able provides a process model account of how problem solvers apply and learn how to better apply principles to solve physics problems. It does not provide as detailed a model of physics problem solving as more recent models (Elio & Scharf, 1990; Ploetzner, 1995), but emphasizes how the order of principle application in formalizable domains can change with practice, and provides a mechanism for predicting the order of principle application that has been matched to extensive amounts of protocol data. It does not model in detail how the principles are used. Novices may use a principle to solve for mass while experts may carry mass forward through the application of a principle knowing that mass will be canceled out later.

At the start of a problem, Able has all the known and unknown variables in its working memory (its top problem solving state). Its problem solving is finished when the target variable or variables are known. Also on the top state are the physics principles. Able has eight principles, such as $F = m\,a$, $F = \mu\,N$, and $x = v_0\,t + 0.5\,a\,t^2$. These equations are represented in a simple way as sets of variables (e.g., F m and a) because Able does not model the lowest level of problem solving—algebraic manipulations—but only models the result that if **F** and **m** are known, then **a** would be known as well.

Figure 1 shows the operators and their relationships. After a problem has been retrieved with FETCH-PROBLEM, problem solving proceeds with a top-level operator proposing to solve the problem. DEVELOP-KNOWLEDGE will later implement single inference steps that directly solve the problem, but initially, nothing can be done, and an impasse is noted by the architecture. In this impasse, the target variable is selected as the variable to solve. APPLY-PRINCIPLE operators are proposed to apply each of the principles on the state. There is some fairly powerful heuristic knowledge included about how to chose an operator that not all problem solvers have (but all Larkin's subjects appear to have had). Those operators that are applying principles that do not have many unknown variables are preferred, but more importantly, operators that propose principles including the target variable are preferred. Principles with the same number of unknowns and relationship to the target are made equivalent. If additional domain knowledge about which principles to select was available, it could apply here as well.

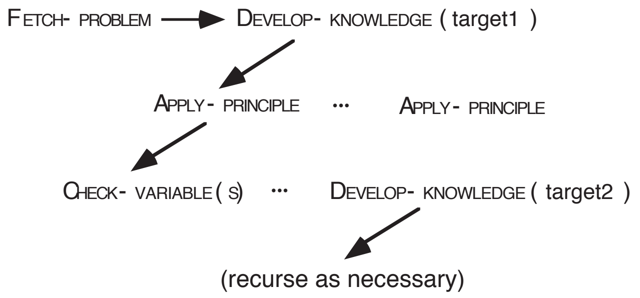


Figure 1: The structure of the operators in Able. Arrows indicate order of application and relationships in the hierarchy. Ellipses (...) indicate that multiple applications of the previous operator may occur.

The implementation of APPLY-PRINCIPLE is not initially available, for it, too, is learned. Another impasse occurs, and lower level CHECK-VARIABLE operators check each of the variables in the principle. If all but the target are known, then the target can be derived, and this is passed up to the higher operator. If variables other than the target are not known, DEVELOP-KNOWLEDGE is applied recursively, with the unknown variable as a target. This leads to backward-chaining behavior that is typical of novices in this domain (Larkin, McDermott, Simon, & Simon, 1980a; Larkin, et al., 1980b).

During problem solving, chunks (new, learned productions) are created that encapsulate the essential aspects of the impasse and the result that was used to resolve the impasse. These chunks allow APPLY-PRINCIPLE to be applied atomically when similar circumstances occur. With additional problem solving, because the bottom-most operators must be learned first, the derivation of unknown variables from known variables eventually occurs with the DEVELOP-KNOWLEDGE operator as well.

Learning changes how Able solves problems. With enough practice, fully learned behavior occurs with DEVELOP-KNOWLEDGE solving problems directly through application of the learned rules, in a forward-chaining way, using the known variables to derive additional known variables. The model changes from being driven in a goal-directed way to apply principles to derive the target variable, to being data-driven, where the known variables are used to directly derive additional known variables.

Practice also drastically speeds up how long Able takes to solve a problem. Able III initially takes 27 model (decision) cycles to solve a typical problem (number 5) on the first attempt. This time includes time to find the principles to apply, to check each of the variables, and recursively solve for variables when this is necessary. After practice over 7 trials with the same problem, Able takes just 2 model cycles and no longer improves. The learning curve that is generated does not fit a power law, but it is difficult to comment further, because there are multiple aspects of the task not yet included in the model.

Able's novice/expert performance characterization is similar in several ways to Klein's (1989) broadly applied theory of recognition-primed decision making (which might more correctly be called "recognition-led problem solving in dynamic tasks"). Like Klein's theory, Able after learning works forward from known information; its behavior is based on previous problem solving; and Able does not consider alternative actions. Able is different in that it is spelled out in enough detail to implement some of the structural details of behavior but in a limited area, whereas Klein's theory remains descriptive.

Able has been applied only to formal domains so far, those "involving a considerable amount of rich semantic knowledge but characterized by a set of principles logically sufficient to solve problems in the domain" (Larkin, 1981, p. 311). Mathematics, physics and sophisticated games (e.g., chess) are formalizable; biology and English literature are much less so. Whether Klein's domains (e.g., of fire fighting) are formal or can be formalized is unclear. The field of cognitive science would assume that they could be, and attempts to build expert systems in these areas are consistent with that belief. Able suggests that it may be possible to create a broad range of cognitive models that start to explain the novice/expert differences that Klein reports by the way they improve through performing tasks.

Able III

Able III is a substantially revised version of Levy's Able-Soar, Jr. (Levy, 1991). In some ways it is a simple revision of an existing model. This view, while true, misses what has been added. In addition to being substantially revised, documented, and extended to cover a few more problems, we have created graphic displays of its generic and specific behaviors, and modified Able's principle application mechanism so that it may directly serve as a building block for other models.

Annotated, shortened and commented

In Able III we have updated Levy's version of Able, originally written in Soar 4, to Soar 7 (Congdon & Laird, 1995). There have been several changes to the Soar architecture in that time, including allowing the reuse of the state structure and removing the explicit problem space structure. Some of the rule syntax, firing and support mechanisms have changed slightly as well.

The relative ease with which Able was translated shows that the basic Soar architecture has not changed much since 1989 in the aspects that Able relies upon. While the functionality has basically stayed the same (Able-Soar, Jr. solved 13 unique dynamics problems, Able III, solves 16), the number of rules has slightly decreased from 52 basic rules (excluding monitoring and problem generating rules) to 47 rules. It is not the case that the rules have become more complicated, for the number of clauses has decreased even more dramatically from 400 to 218. The differences in these rule sets suggest that the syntax for specifying models in Soar has become simpler without substantially changing the architecture, which is indeed what its architects have endeavored to do (Laird, Huffman, & Portelli, 1990).

Able was not fundamentally affected by the changes in the architecture in the last five years. One of the valid criticisms suggested by Cooper and Shallice (1995) was that as the Soar architecture was modified, older models must be carried forward for their results to remain valid. This has not typically happened each time the architecture has been released as new software. The Soar community has not been convinced of the need because they understood the changes, and theoretically the changes have nearly always been small with limited impact on existing models. Able is a relatively straightforward model, but the absence of problems suggests that the approach Cooper and Shallice put forward did not correctly classify changes, and the changes they have typically noted are implementation choices rather then changes in the theory. More complicated models, however, have a greater chance of suffering from architectural changes.

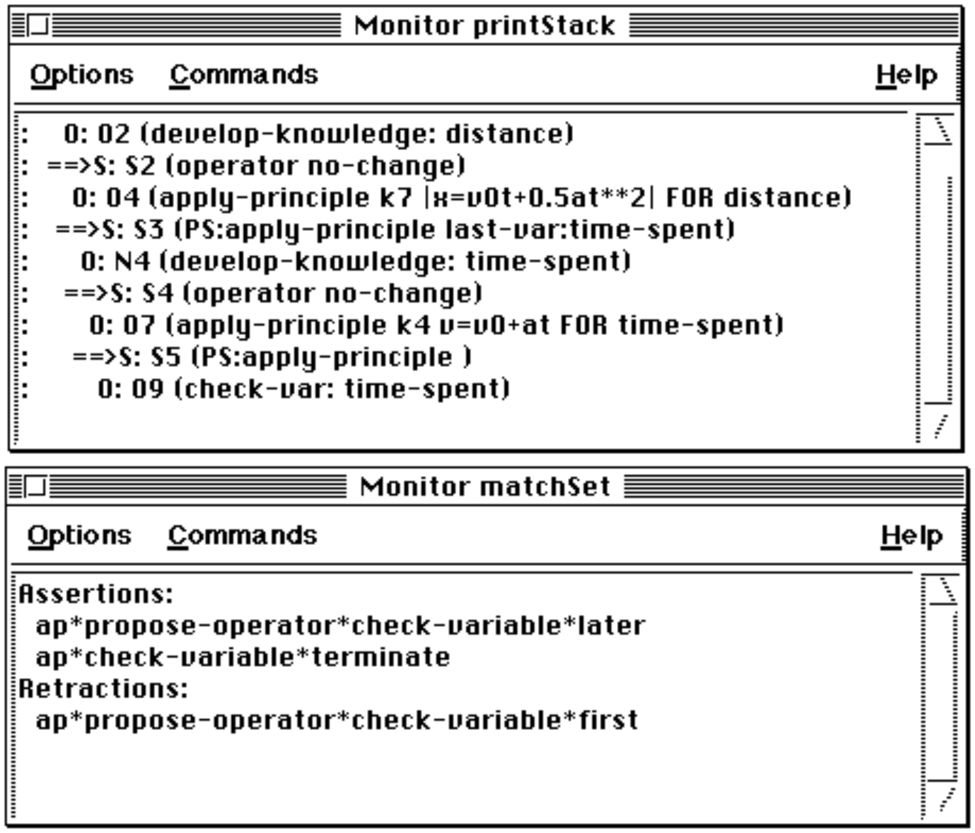
Visual interface makes behavior visible

Graphic displays are often useful aids when problem solving (Larkin & Simon, 1987). For a brief period, when Soar was implemented in Lisp, a set of general graphic displays were available (Ritter & Larkin, 1994). These led to some new understandings about Soar models, for example, that few extant models did extensive search *in* problem spaces, but instead generally used relatively few operators in each space. When Soar was reimplemented in C, these graphic displays were left behind. By including Tcl/Tk (Ousterhout, 1994) in Soar, these types of displays can be recreated.

We have created two types of displays for working with Able III. Some of these will be useful for developing and explaining nearly any Soar model, and some specifically when working with Able III. They provide examples of displays that would be useful for any cognitive architecture.

General Architecture Displays We have built with Randy Jones a set of general displays for working with any Soar model called the Tcl/Tk Soar Interface (TSI) and include them with the Able model release. There is an interaction window (not shown here), which allows the user to interact with Soar on a basic level using menus. It is similar to, but much simpler than, the Soar Development Environment (Hucka, 1994) within GNU-Emacs. There are also three displays that can be updated every model cycle to display the current goal stack, the rules that will fire next when the model is run, and the details on how the next operator will be selected. In each of these displays the user can bring up a help menu and directly run the simulation. The ten most commonly used Soar commands (Nichols & Ritter, 1995) are supported with the displays, or bound to keystrokes or menus. These displays currently work with Soar on the Mac and under Unix, and offer a full set or a restricted set of commands (for teaching novices).

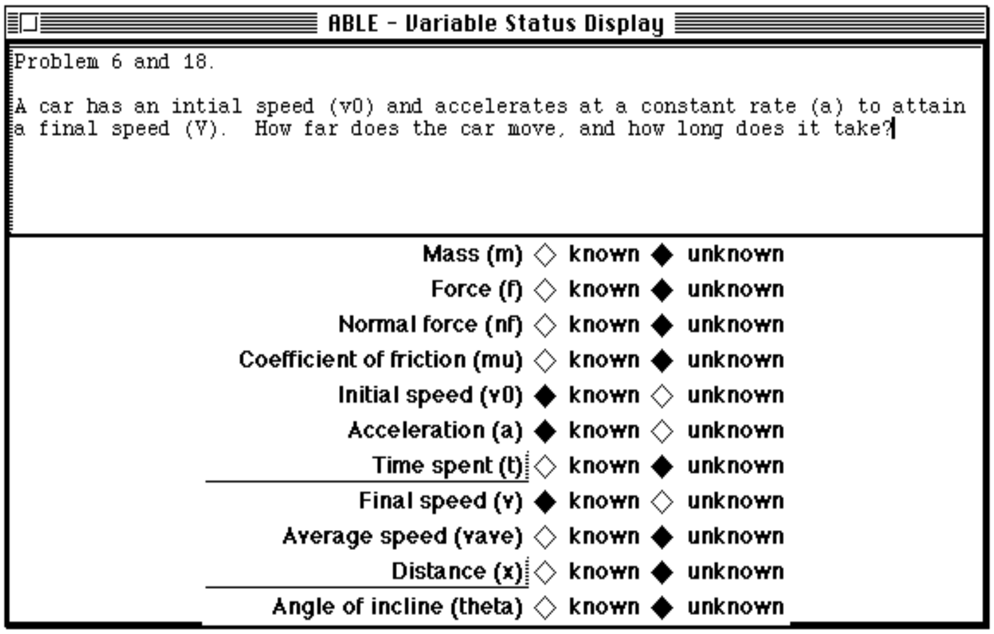
In U. Schmid, J. Krems, & F. Wysotzki (Eds.), *Proceedings of the First European Workshop on Cognitive Modeling*. 22-30. Berlin: Forschungsberichte des Fachbereichs Informatik, Technische Universität Berlin.



```
:   O: O2 (develop-knowledge: distance)
:  ==>S: S2 (operator no-change)
:    O: O4 (apply-principle k7 |x=v0t+0.5at**2| FOR distance)
:   ==>S: S3 (PS:apply-principle last-var:time-spent)
:     O: N4 (develop-knowledge: time-spent)
:    ==>S: S4 (operator no-change)
:      O: O7 (apply-principle k4 v=v0+at FOR time-spent)
:     ==>S: S5 (PS:apply-principle )
:       O: O9 (check-var: time-spent)
```

```
Assertions:
 ap*propose-operator*check-variable*later
 ap*check-variable*terminate
Retractions:
 ap*propose-operator*check-variable*first
```

Figure 2: The general TSI display windows included with Able III, showing the current goal stack (top window), and the rules that will fire next (bottom window).



```
Problem 6 and 18.

A car has an intial speed (v0) and accelerates at a constant rate (a) to attain
a final speed (V).  How far does the car move, and how long does it take?
```

Mass (m)	◇	known	◆	unknown
Force (f)	◇	known	◆	unknown
Normal force (nf)	◇	known	◆	unknown
Coefficient of friction (mu)	◇	known	◆	unknown
Initial speed (v0)	◆	known	◇	unknown
Acceleration (a)	◆	known	◇	unknown
Time spent (t)	◇	known	◆	unknown
Final speed (v)	◆	known	◇	unknown
Average speed (vave)	◇	known	◆	unknown
Distance (x)	◇	known	◆	unknown
Angle of incline (theta)	◇	known	◆	unknown

Figure 3: The problem display in Able, showing the problem (as text in the top pane) and the current status (known/unknown) of the variables. The target variables, Time spent (t) and Distance (x), are in raised text on the screen, which appear here as underlined text.

The continuous goal stack display, shown at the top of Figure 2, indicates the order of operator applications and the current goal stack. Users can examine the substructure of objects in the stack in a separate window (not shown). Users can also select how much detail is displayed, choosing to print several layers of substructure by default, or continue to examine substructures level by level.

The continuous match set display, shown at the bottom of Figure 2**,** provides a display of the rules that have matched the current working memory, and will fire in the next model cycle. Users can print them in a separate window for inspection.

Only anecdotal evidence about the usefulness of these displays is available so far. When an early version was introduced to a psychology class on programming cognitive models, all of the students chose to use the displays (whereas they could not all use Emacs). We will be using the latest version of the displays this spring in the same class.

Application Specific Displays There are two displays included with Able to help explain how it works and display its behavior. The first display, shown in Figure 3, describes the problem Able is working on. It includes some text explaining the story problem associated with the physics problem, and the current status of the variables, that is, known or unknown. It also indicates which variables are targets. This display currently only works with the physics variables in Able.

The second display, shown in Figure 4, indicates the order of principle application. It shows that the Able model when it is a novice (really, an apprentice, since it knows something) works backward from the target variable. The more expert Able, after it has solved problems and has nearly doubled the number of rules it has, does not appear to apply principles at all, but works forward immediately deriving what is known. This display is based on the application of principles, so it will work with any set of principles represented within this framework.

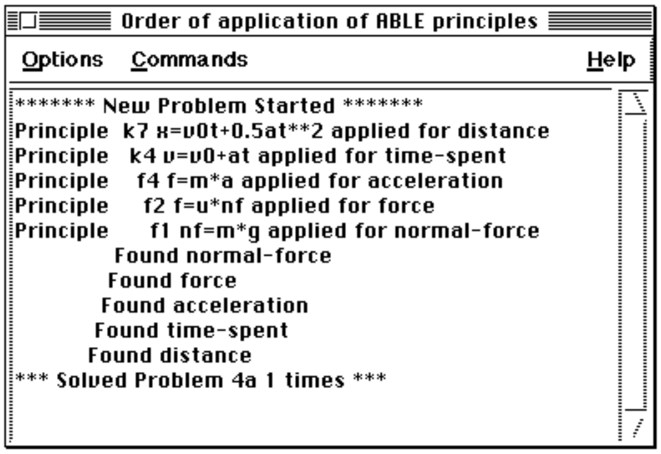
Using principles as higher level language

The previous work with Able did not treat its principles and their application mechanism as a high level programming language for cognitive models, but they can be seen that way. This mechanism is useful enough that it should be available as a general utility, and based on our work it is now a straightforward task to add new principles for another domain.

New problems can be included by representing their features on the top state. New principles are represented one per production rule. The principle application mechanism in Able can then solve the problem. Additional knowledge can be added, but the weak methods of search in Soar and Able will otherwise solve the problem if it is solvable.

This mechanism could be used to model novice-expert transitions in other domains, and provide a way to include routine learning in models. With any set of principles, initial behavior of the model to choose and apply the principles will be effortful and susceptible to dead-ends. With practice, the model's performance will become situation driven and faster. This approach may make it easier to create models in Soar by providing a mechanism that more closely approximates the highest conceptual level, the knowledge level (Newell, 1982).

To test how easy it would be to create a new model, we created and tested in 30 minutes a model that solved a gas physics problem noted as one that should show expert/novice differences (vanSomeren, Barnard, & Sandberg, 1994, p. 14-15) . The model consists of three production rules to be added to the existing Able mechanism; a simple model for a simple problem, but it demonstrates that models can be created quickly.



```
******* New Problem Started *******
Principle  k7 x=v0t+0.5at**2 applied for distance
Principle  k4 v=v0+at applied for time-spent
Principle   f4 f=m*a applied for acceleration
Principle    f2 f=u*nf applied for force
Principle     f1 nf=m*g applied for normal-force
              Found normal-force
             Found force
            Found acceleration
           Found time-spent
          Found distance
*** Solved Problem 4a 1 times ***
```

Figure 4: The Application of principles display in Able shows the order that principles are applied. With practice on this problem, explicit reference to principles disappears.

Problems remain with using Able as a utility, however. It was developed to model behavior in formal domains. Not all domains are formal. It models the novice-expert transition in well under 100 trials, which normally takes years of practice. The transition that is modeled, the order in which to apply principles, may be learned this quickly, but then the model is not modeling the gamut of knowledge that makes up an expert. The principle application mechanism is also unrealistic in the way it uses working memory. It keeps the problem and all the principles on the top state, which is not appropriate. These flaws should not be taken as reasons to reject it, but rather clear indications about where it can be improved.

Summary

Able III as an exemplar suggests two useful additions to cognitive modeling and notes an open problem. The first addition that Able suggests should be routinely included is to provide graphic displays that make models' behavior easier to understand. Several of these displays will be useful when developing any Soar model because they make the internal behavior visibly explicit. The utility of the Able specific displays also suggest that similar displays should be provided for other models in other environments as well.

The other addition to cognitive modeling that Able III proposes is the explicit need to abstract and export the fundamental mechanism for inclusion in other models, even when working within a cognitive architecture. Here, the principle application mechanism becomes a utility as a new programming language. This is an important exemplar, for cognitive models as sets of knowledge should be reusable, including their knowledge based mechanisms.

These displays have told us something about Soar as well. Soar proposes that there are three interesting levels of theoretical interest: the knowledge level, the problem space level, and the symbolic or implementation level (Newell, 1990). Implementing these displays have emphasized that the problem space level does not explicitly exist in the code that makes up Soar models—it is an emergent behavior arising from production firings. There are commands to manipulate productions, for example, to delete them. There are far fewer existing user commands to manipulate objects on the problem space level. Creating a visual interface helps compare these levels and encourages us to ask new questions about the models. While we now see on the general display menus the ability to list productions and their firing counts, we will have to extend the system in order to list the operators and how often they have been used. We have already extended the system to allow users to insert operators, and this seems quite useful.

How best to document and explain the model's code remains an open problem. We have taken some care to document and explain the model as a program. Good practice in this area has not fundamentally changed in the last five years. That is, there are no established standards and no well accepted best way to explain the model as productions. With Richard Young, we have tried creating "illuminated code", which includes embedded HTML commands to illustrate code and provide additional information (an example is available in http://www.psyc.nott.ac.uk/users/ritter/pst/analogy/answers/anal-ans4.html). Having illuminated code was, in the end, quite useful for teaching, but in our limited experience it is not good for writing and extending live code.

We believe that for cognitive modeling to succeed, not just survive, more models will have to be prepared in this style. Models must become easy to understood, easy to extend, and easy to reuse. Packaging programs as utilities with displays provides one way of facilitating this.

Acknowledgments

Initial versions of some of the graphic displays were created by Karl Schwamb, and the common graphic displays were written with Randy Jones. Moritz Baumann, Pete Bibby, and Rolf Ploetzner provided useful comments. Support for this work has been provided by the DRA, contract number 2024/004, and by the ESRC Centre for Research in Development, Instruction and Training. Any errors remain the fault of the authors, and an acknowledgment here does not indicate an endorsement of this report from those listed. This paper is a revised and extended version of a paper presented at the First European Cognitive Modeling Workshop.

References

Bass, E. J., Baxter, G. D., & Ritter, F. E. (1995). Using cognitive models to control simulations of complex systems. *AISB Quarterly, 93*, 18-25.

Congdon, C. B., & Laird, J. E. (1995). The Soar user's manual, version 7. Ann Arbor, MI: Electrical Engineering and Computer Science Department, U. of Michigan.

Cooper, R., & Shallice, T. (1995). Soar and the case for unified theories of cognition. *Cognition, 55*, 115-149.

Elio, R., & Scharf, P. B. (1990). Modeling novice-to-expert shifts in problem-solving strategy and knowledge organization. *Cognitive Science, 14*, 579-639.

Hucka, M. (1994). *The Soar Development Environment*. Ann Arbor, MI: Artificial Intelligence Laboratory, U. of Michigan. Also available through http://www.cs.cmu.edu/afs/cs/project/soar/www/soar-archive-software.html.

Klein, G. A. (1989). Recognition-primed decisions. In W. B. Rouse (Ed.), *Advances in man-machine systems research (vol. 5)*. Greenwich, CT: JAI.

Laird, J., Huffman, S., & Portelli, M. (1990). Status of NNPSCM and S-support. In T. Johnson (Ed.), *Thirteenth Soar Workshop*. 49-51. THE Ohio State University: The Soar Group.

Larkin, J. H. (1981). Enriching formal knowledge: A model for learning to solve textbook physics problems. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition*. Hillsdale, NJ: LEA.

Larkin, J. H., McDermott, J., Simon, D. P., & Simon, H. A. (1980a). Expert and novice performance in solving physics problems. *Science, 208*, 1335-1342.

Larkin, J. H., McDermott, J., Simon, D. P., & Simon, H. A. (1980b). Models of competence in solving physics problems. *Cognitive Science, 4*, 317-345.

Larkin, J. H., & Simon, H. A. (1981). Learning through growth of skill in mental modeling. In H. A. Simon (Ed.), *Models of thought II*. New Haven, CT: Yale University Press.

Larkin, J. H., & Simon, H. A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science, 11*(1), 65-99.

Levy, B. (1991). Able Soar Jr: A model for learning to solve kinematic problems. Unpublished.

Newell, A. (1982). The knowledge level. *Artificial Intelligence, 18*, 87-127.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Nichols, S., & Ritter, F. E. (1995). A theoretically motivated tool for automatically generating command aliases. In *Proceedings of the CHI '95 Conference on Human Factors in Computer Systems*. 393-400. New York, NY: ACM.

Ousterhout, J. K. (1994). *Tcl and the Tk toolkit*. Reading, MA: Addison-Wesley.

Ploetzner, R. (1995). The construction of coordination of complementary problem representations in physics. *J. of Artificial Intelligence in Education, 6*(2/3), 203-238.

Ritter, F. E., & Larkin, J. H. (1994). Using process models to summarize sequences of human actions. *Human-Computer Interaction, 9*(3&4), 345-383.

vanSomeren, M. W., Barnard, Y. F., & Sandberg, J. A. C. (1994). *The Think Aloud Method: A practical guide to modelling cognitive processes*. London/San Diego: Academic Press.