Chapter 9

Contributions and steps toward the vision of routine automatic model testing

Compared with Chapter 1, we are not in the same place in many ways, and we are considerably further along toward the capacity to perform routine process model testing. Progress has been made on defining a methodology for testing the sequential predictions of process models. A computer environment has been implemented to support this methodology, and this environment has been used to test an actual model with actual data. Portions of the environment are used by researchers around the world. The environment was used to test and extend the sequentiality assumption of Ericsson and Simon's (1984) theory of verbal protocol production. The path to an intelligent automatic modeling system based on agent tracking is clearer. Only model testing (open analysis) has been considered in this work, but the methodology and environment should largely be applicable to using models to classify sequential behavior (closed analysis) for such things as cognitive-based testing (Ohlsson, 1990).

The central problem: dealing with large amounts of information. Within the methodology of TBPA the essential problem in testing process models still appears to be one of manipulating and understanding the large amounts of information involved: the model, its predictions, and the data used to test it. Scientists do not decry the difficulty of model creation and manipulation as often as they have the amount of bookkeeping required for testing the sequential predictions. The size of the data sets prove a real problem; the amount of qualitative information used in this task is relatively large given the analyst's limited processing capabilities.

Each of the steps in TBPA requires manipulating large amounts of information. This is a central problem that runs through this work, and it is fought in every tool in the Soar/MT environment. Two approaches have been developed for dealing with it. The first is to automate as many tasks as possible, and to support the analyst for the remainder. The second is to design and use visual displays of information.

Secret weapon #1: Automate and support. Automating aspects of each step reduces the work load required of the analyst. Soar/MT assists the analyst by automatically aligning unambiguous parts of protocols, creating model-based summary displays of the comparison, and providing many aids for displaying and manipulating the model. Although the automatic processes fall short of the ideal speed, and still must be speeded up through better algorithm and data structure design, they have proved useful in their current state. The process is not so inherently large or computationally intensive that so-called super-computing will be required.

The data set presented with Browser-Soar (Peck & John, 1992) is not the largest data set ever used to test a model (although it is fairly large, see Table 2-2), but Soar/MT has substantially speeded up the analysis of this data set. We can now imagine analyzing enough protocol data to achieve Ericsson and Simon's (1984) vision of verbal reports as data.

Supporting the analyst in performing the tasks that are not yet automated has required careful design of the displays and manipulation tools for the large amount of information. The current maximum size of the predictions and data, not including the model, is about 330 Kb. The analyst cannot directly visualize and manipulate information sets the size of a small phone book (5,000 names at 60 bits per name, or 300 Kb total). Special displays have been created to show the important trends in the data, which is the next secret weapon.

Secret weapon #2: Scientific visualization of qualitative information. Appropriate visual displays can support faster processing rates and provide new insights (Larkin & Simon, 1981). Visual displays of qualitative information have become central to quantitative data analysis in many domains and they have lead to the major methodology of scientific visualization.

Visual displays should now be considered essential for performing each step of protocol analysis and process model testing. Visual displays help the analyst understand the model's structure and performance, relating them to each other in a single display, the SX graphic display. Tabular displays of the model's predictions, the data, and their correspondences show simple and directly where the model's predictions do and do not match the data. Other displays aggregate the correspondences in terms of the model components and in terms of relative processing rates. These displays summarize where the model performs well and where it performs poorly, providing clues about where and how to improve the model's fit to the data.

9.1 A methodology for testing the sequential predictions of process models

Trace Based Protocol Analysis (TBPA), a methodology for testing the sequential predictions of process models with protocol data has been defined through listing its inputs, processing steps, and their requirements. TBPA tests a model by running it to generate a trace of how the model performs the task. This trace provides a set of theoretical predictions of what will be found in a subject's verbal and non-verbal protocol, and it is used to interpret the data. TBPA is designed to be an integrated and iterative process, so a summary of where the predictions are unmatched in the protocol is then used to modify the model, and the model is run again. The necessary inputs to TBPA, its steps, and the processing requirements for each step to perform the testing routinely, were specified in enough detail to create a computer environment to support this methodology.

Clarification of the testing process. What it means to test the model became clearer from specifying each step in the process. What are tested in any given episode are the model's predictions. The comparison of the predictions with the data is not just one of alignment. The model's predictions are used to interpret the data. With unambiguous data, such as mouse clicks on menu items, the process appears to be one of simple alignment and it can be treated that way. When the data are verbal protocols, then the items in the trace may provide substantial guidance for interpreting the meaning and function of the information described verbally.

Some theories require every prediction to be matched, but the theory of verbal protocol used to interpret the utterances (Ericsson & Simon, 1984) states that not every possible prediction will be found. The model's predictions are predictions of what could be found in the subject's verbal protocol.

The need for declarative versions of models. It is necessary for model based analysis to refer to structures of the model and to note which parts of the model did and did not apply, or were and were not supported. It is necessary to have declarative representations of process models representing procedural knowledge. Running the model to create the structures upon demand is not enough. There is the simple problem that the structures will be created and then disappear as the context changes. There is also a more complicated problem of coverage, on any given run not all the possible structures will be created. Examining the initial implementation of the model is not adequate either, the model might learn from its environment, and computing all the model's structures is equivalent to running it.

At a minimum, it is necessary to create a description of the model computed by observing the model's performance over time, although combinations of the other methods, such as derivation from the static structure, are a useful adjunct. Although this method is the best way to build the model, even this model is not guaranteed to be complete.

The DSI creates a declarative representation of Soar models. While the Soar model runs, the DSI displays and remembers which and how often the problem spaces, states, and operators have been applied. By watching the model as it runs the DSI builds up as complete a view of the model as is possible. The resulting description can be used by other components in the environment. The interpretation environment can use it to initially code the data. The saved model can be used to summarize the correspondences created though interpreting and aligning the data with respect to the predictions.

9.2 Each step in the methodology was supported in a software environment

An environment to support an analyst performing TBPA has been created based on its requirements. The environment directly supports the main tasks of model tracing; interpreting and aligning the model's predictions with the data, both automatically and semi-automatically; aggregating the comparison data in a variety of displays designed to show how to improve the model; understanding and modifying the model based on how it does not fit.

The steps were specified and broken down to a level that they could be performed automatically, or semi-automatically. Building, loading, and running models was supported in a semi-automatic way. Many small tasks are supported through keystroke macros in the structured editors and smarter interfaces. Finding the emergent properties of Soar models (listing the problem spaces and their operators) is supported, as is counting how often they are instantiated. Unambiguous portions of the subject data are now matched automatically. The same algorithm can be used to interpret and align the data in an incomplete and heuristic fashion, requiring the analyst only to check and clean up the approximate interpretation. Finally, the analytic displays can be automatically created from the comparison data.

The environment also supports the requirements of integrating the steps, automating the tasks where possible, and supporting the analyst for the rest. The environment and the methodology it supports were tested by testing a process model, and in the process learning new things about the model and its fit to the data. The tasks in TBPA that the environment support overlap with other tasks often performed in cognitive model building and modification, data manipulation with a tabular display, and exploratory data analysis.

Sub-portions of the environment supported other users doing the sub-tasks for different reasons, the DSI for AI modeling, Dismal for spreadsheets, and S-mode for statistics and graphing. A survey of users of the DSI found that over half the Soar community uses some portion of the DSI whenever they use Soar. It would be safe to say that pieces of the environment supporting these tasks are in use by over 500 researchers around the world.

The analyses are fast enough to be considered routine. A minute long episode of subject data (approximately 20 verbal segments and 30 motor actions in the browsing task) can now be compared with the model's predictions in 2.5 hours given sufficient inputs, the process model and transcribed data. This is almost within automating range; when it took 60 hours to perform (estimate derived from Ohlsson, 1980), too many under specified processes were required, and automating this task was not conceivable.

Example testing of Browser-Soar using TBPA. The methodology was demonstrated on the Browser-Soar (Peck & John, 1992) model. A set of suggestions for improving Browser-Soar was generated, and one of them was implemented. This lead to a slightly better fit, but more importantly, to a much more parsimonious model. Browser-Soar and its data set did not push this methodology in all directions, but this was good. It allowed making headway on some problems by avoiding others.

9.2.1 Interpreting and aligning the model's predictions and the data

This thesis explored the automatic alignment of unambiguous data to model predictions. The Card algorithm for doing this was slightly improved, and its behavior characterized more clearly.

A spreadsheet approach to the comparison process was demonstrated, and it appears to visually support many of the necessary operations on the data that would otherwise require extensive computation by hand. For example, areas where the predictions match the data in a denser manner is clearly presented. The spreadsheet was also effective in supporting the analyst in easily adjusting the alignment manually when necessary.

9.2.2 Analyzing the results of the testing process

A lack of clarity about what measures are necessary or desirable for measuring predictions fit to the data may have contributed to the lack of progress. The review in Chapter 2 outlined the uses and abuses of several of these measures, and championed Grant's (1962) approach of analytic testing, of finding out where the model can be improved.

A display for showing the support of operators in the model was automated, and an additional family of displays were produced for presenting and analyzing the relative processing rate of the subject with respect to the model. These two sets of displays can be created automatically from the comparison data. They have shown the periodicity of human browsing behavior, the types of mismatches between model and data, and ways to improve the fit of the model. There are many ways for data to not match the model. Additional graphs will be necessary, so an environment is provided to assist in editing and designing these graphs.

9.2.3 Steps related to manipulating the model: Prediction generation and modification

While the model's components are used throughout the analyses, the process model itself is directly involved in two steps, that of generating the sequential predictions, and the final step of revising the model based on the testing process.

Generating the predictions. Generating the model's predictions in a way that they can be used for automatic alignment has required extending infrastructure from the model (in this case, a Soar model) out further toward the data. This has resulted in a better trace — one that is less ambiguous and more readable by humans. Based on the example analysis, we also found that a problem space model must provide state traces in addition to operator traces.

The improved trace lead to an unexpected benefit. We found that deriving aggregate measures in the trace was useful for comparing models and describing their behavior in general terms.

Manipulating and creating models. The Developmental Soar Interface demonstrates the feasibility and utility of several design principles. Across the environment it was possible to meet the design shown in Table 9-34.

Table 9-34: The ease of use and learnability design features met by each tool in the environment.

- Provide a path to expertise through:
  - Menus to drive the interface.
  - Keystroke accelerators available and automatically placed on menus for users to learn.
  - Help provided for each command on request.
  - Hardcopy manuals also available on-line through the menu.
- Treat structures on the theoretical level as first class objects.
- Provide a general tool with macro facilities.

These features make the task of inserting the model's knowledge into Soar easier. Keystroke level models can be presented as evidence for this, as well as the fact that approximately two-thirds of the Soar community now use some portion of the DSI in their daily work.

Node based graph display. Many structure display algorithms draw the complete structure, forcing the user to scroll a window pane across it. Presenting Soar's working memory contents is such a structure

display task. The set of tasks users need to perform when examining the structures within working memory have been identified, and a display meeting these requirements has been designed and implemented. The task analysis lead to a different design than a big scrollable window — a node-based design that allows users to open up individually selected nodes in working memory, close their parents, and so on. The users seem pleased, and it provides a much faster display.

General results about Soar. The visual and structural representations in the Developmental Soar Interface highlighted several features of Soar models and the TAQL macro language. For TAQL, the templates within the structured editor provided a measure of the cumbersome size of the TAQL syntax.

For several specific models we were able to display how their behavior is not best characterized as just search in problem spaces. Behavior within many models now includes routine behavior, search through problem spaces, migration of knowledge between problem spaces, and composition of knowledge.

Within Soar models in general, displaying their behavior graphically pointed out how ephemeral problem spaces and their structures are. In many ways the application and interactions of objects on the problem space level should be considered as emergent behavior. The structure of the model is only available from repeated viewing; the model itself has no representation of itself, and cannot conjure up all the problem spaces and operators that are possible.

9.2.4 The synergy from integration

The environment receives much of its power from integration. The model, its behavior, the subject data, and the comparison of the model and the data all exist in the same environment. This supports several analyses that would be difficult without the integration and it allows them to be much more fluid. Integration allows: (a) direct, preliminary coding of the protocols based on the model's components; (b) appropriate mixed (text and symbolic graphics) presentation of data in the DSI; (c) appropriate mixed (text and symbolic graphics) presentation of data in the analyses; and (d) the portions of the trace that were well aligned and not well aligned could be directly compared with the model's structures.

9.3 Validated and extended the sequentiality assumption of protocol generation theory

Using the TBPA methodology and the Soar/MT environment, the Browser-Soar model and data of Peck & John (1992) were re-examined. Besides providing a test-bed for the methodology and environment, this effort yielded the following new scientific result.

The verbal protocol production theory of Ericsson and Simon (1984) assumes that working memory structures are reported in the order that they enter working memory. This assumption can be tested with a model that predicts when objects enter working memory. The Soar/MT display of the relative processing rates of the Browser-Soar model and the subject provided a direct visual test of this assumption. The underlying data structures were then directly queried to confirm and count the number of sequential and non-sequential pairs of events there were. In every episode of the Browser-Soar, the sequentiality assumption was found to hold for the verbal protocol. An examination of the non-verbal protocol segments found that they too were always performed in the same order as the model, both for overt task actions, and for actions that were not directly related to the task, such as moving the mouse pointer over words being read on the screen.

The two data streams appeared to be presented in a non-sequential order. Verbal utterances typically lagged 10 to 30 simulation cycles (approximately 1 to 3 s) behind the overt actions; and rarely (3/300) they lagged up to 400 simulation cycles (approximately 40 s).

The shorter lags were probably reports of working memory delayed by workload associated with the task, and minor inconsistencies in the model. Examination of the correspondences showed that the

primary cause of the long lags was probably an artifact of the interpretation process. The verbal utterances in the analysis were matched to operators rather than to the state information created by the operators. This approximation simplified the analysis considerably, and it should remain available — it is a valuable technique. But it must be seen as only an approximation; one that will sometimes lead to inconsistencies in the comparison. Any operator that sets up long lasting state information can cause this problem.

As a result of these analyses it is proposed that the sequentiality assumption holds for both verbal utteranances and task actions. Including motor task actions as part of the protocol provides reference points for fixing the correspondences between the predictions and subject's actions, and allows the lag of the verbal utterances to be measured.

9.4 Progress toward the vision of routine applied theoretically guided protocol analysis

This work has made appreciable progress toward the vision of automatic modeling. All the parts of Soar/MT are part of a grand vision of what an integrated modeling and data analysis system would need to do, and could do. The major steps and inputs have been identified as the parts of TBPA, and a prototype environment has been created that an automatic modeling system would need. The next steps will be to create initial models, and to provide a more intelligent process for interpreting ambiguous data with respect to the model's predictions.

Because this environment is based on an architecture for general intelligence, it is conceptually possible to add knowledge to the architecture of how to perform parts or all of the analysis. To do this completely would require incorporating a complete model of the analyst. However, the architecture used in this environment, Soar, also learns. So perhaps an easier, but less direct way to automate this task might be through having a Soar-based agent learn to perform the analyses by watching a series of analyses. As it watched a series of routine analyses over similar episodes be performed, it could follow along, learning how to run the analyses, and then driving the analyses programs itself.

Not that we are there, but we can now see further down the path toward completely automatic modeling. If NL-Soar (a Soar system for interpreting natural language) were to be incorporated, then Soar/MT might take in instructions for different experiments, and use the models that NL-Soar creates from reading the instructions as initial models to predict the behavior of subjects for each experiment (Lewis, Newell & Polk, 1989; Newell, 1991). The alignment also could be automated. The non-verbal overt actions can be compared directly; the verbal utterances would have data structures, the predictions, laying around that are designed to be sufficient to parse them. NL-Soar (Lehman et al., 1991) is available as a potential parser designed to use these predictions.

This style of protocol analysis requires further computer science and AI work: performing the alignment of predictions to natural language, running the models more quickly, and gathering better statistics. But it remains a task within psychology: the real use is for comparing protocols against models' predictions.

Remaining problems. Many problems remained in this methodology and environment. I would like to note a few here to admit its deficiencies, to warn potential users of the current specificity of the tasks Soar/MT can address, and to suggest directions for future work.

How to aggregate support from the predictions to the model structures is not always as straightforward as it appeared in the sample analysis of Browser-Soar. There is a problem of specifying how the predictions are used to interpret the data. There is also a problem in specifying how to aggregate support for model components. Across episodes, the structures in the model that generated the predictions remain and summarize the behavior over time. The current model implemented its operators rather directly and in the same manner each time. This need not be the case. Consider an *Add* operator such as Siegler uses in his work modeling children's arithmetic knowledge (Siegler,

1988; Siegler & Shrager, 1984). Different operands result in different reaction times and error patterns. Assigning support to an operator in this case must be differentiated by the operator's arguments, and a representation for this must be developed. So there is an additional step to TBPA, not yet made explicit, of translating the support that individual predictions receive from the data back to the structures in the model that generated them.

The analyst is currently left with an abduction task of improving the fit with indications of where the model does not fit and with tools for understanding and modifying the model. There are some simple rules that would apply in specific circumstances, and these were noted in the chapter describing the graphical measures of model fit. The possibility of finding a more complete and algorithmic description, like Heise (1987, 1989; Corsaro & Heise, 1990; Heise & Lewis, 1991) provides for his models, should be explored.

Speed, always and everywhere — the analyst always desires a faster system that performs more complicated analyses automatically. Partial views of the data and model are included in this. The recent translation of Soar to the C language offers a speedup in the basic architecture. Taking advantage of this may require translating the DSI into C.

Directions for future work. The way to improve this methodology is the same way to improve a model, by testing and using it on additional models and data sets. Some preliminary discussions have taken place with other researchers about using Soar/MT to test their process models, usually models implemented in Soar.

The software environment could be automated further, and as noted in Chapter 3, the next direct step toward automatic agent modeling would be to represent the knowledge to perform a single step as a Soar model. This would provide further automation. One of the potential places for doing this would be to have NL-Soar parse the verbal utterances, another would be to further automate the generation of the analytical diagrams.

9.5 Concluding remarks

We build our theories, test them, then modify them, iterating through a loop. This loop was described briefly and perhaps for the first time with respect to process models and protocol analysis by Feldman (1962, p. 342). But not surprisingly, it is like all theory testing in science. Models are not primarily tested to be rejected (as the popularization of Popper's (1959) views goes), or tested simply with a significance test to determine their value, but models are tested in order to improve them (Grant, 1962; Newell, 1990, p. 14). By using protocols to test these models, we are not attempting to *code* a segment so that it is *coded*, but we are using the data to build a model (e.g., a simulation process model). That is, to test whether subjects perform the same actions in the same order as the model predicts.

Because they will allow us to see new things, new analyses and tools are also science (Hall, 1992; Laird & Rosenbloom, 1992; Newell, 1991; Ohlsson, 1990; Simon, 1991). New scientific problems are found this way (Toulmin, 1972). Indeed, much of what science consists of — what is passed on from generation to generation of scientists — is just technique (Ohlsson, 1990; Toulmin, 1972).

Because of the difficulties associated with creating process models and of manipulating protocol data, sometimes analysts have lost sight of this fundamental nature of protocol analysis. The technique of testing process models' predictions of sequential behavior has been nudged forward just a bit.