A methodology and software environment for testing process model's sequential predictions with protocols

Frank E. Ritter

20 December 1992
CMU-CS-93-101

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted to the Carnegie-Mellon University Department of Psychology in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Psychology in the AI and Psychology program

This research was partially sponsored by a training grant from the Air Force Office of Scientific Research, Bolling AFB, DC; in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U. S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597, and in part by the School of Computer Science, Carnegie-Mellon University. The research was also supported in part by Digital Equipment Corporation through an equipment grant. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of U. S. Government or Digital Equipment Corporation.

Chapter 2

Testing process models with protocol data: Review of past work

This chapter reviews the previous uses of protocol data for building models and testing them, the various tools that have been built to assist in this, and the measures of model fit to sequential data that have either been computed by hand or incorporated into tools. The lessons noted here will be useful guides for the methodology for testing process models developed in the next chapter.

Process model testing does not live alone, but on a continuum along with data analysis, model generation, and model refinement. These activities can be viewed as manipulating two information streams, the subject's action sequences and the model's action sequences. The various uses of protocol data can be compared with respect to the strength of the model being developed, and how it uses the two information streams. With a more detailed description of this process we can explain the apparent difficulty of using protocols in general, and for testing process models in particular.

Surveying the previous tools for manipulating and comparing protocols to a model indicates several general needs. On the data side, these include the ability to edit the protocols on the segment level, to view large numbers of segments simultaneously, and to compute aggregate measures of them. Similar tools are required of the model, but are less often provided. The structure of the model must be available, and its performance must be aggregated if it is to be compared with aggregate subject measures. The correspondence between the model's predictions and the data must be computed and viewable. It is very enlightening to be able to directly compare these aggregate measures with the model. Similar tools are required for gathering and manipulating the model's behavior, but they are less often available.

Examining the previous measures of model fit indicates several measures that should be included in any future tool. Being able to aggregate the model's behavior in order to describe it and compare it with the subjects behavior is also important. This cannot be directly incorporated into any tool, but must be indirectly supported.

2.1 The possible relationships between process models and protocols

This section creates a framework for organizing most uses of protocol data with respect to process models. The use of protocols are shown to exist on a continuum from model building to model testing. The comparison between the model's predictions and the protocol data can be described as comparing two information streams at various levels of resolution. Examining these previous uses of protocol data provides several guidelines for creating tools for testing process models with protocol data.

Figure 2-2 shows an information stream description of the sequential data structures, analysis, and possible data comparisons between process models and subject data. These operations and information streams are a superset of the data and operations for less predictive models so they provide a conceptual framework for all analyses of action sequences in terms of the information being manipulated, its transformation and losses, and aggregation processes. While only a single stream is represented in the diagram, each virtual stream can be made up of several streams. For example, the human information stream may contain separate streams for verbal protocols, eye movement protocols, and key presses.

The desired end of architecture + knowledge. When developing a process model the analyst starts in the upper left corner of Figure 2-2 with a *mind + knowledge*. We believe that the most desirable analysis is to find out what is in that box, that is, what is necessary to recreate the mind's behavior? What is the *architecture* of the mind, and what *knowledge* must it have to do a particular task? We attempt to duplicate this box by creating a process model that is also broken up into an *architecture* to interpret and apply specific *knowledge* to perform the given task (Newell, 1990; Newell, 1992). The knowledge represents the information required to perform the particular task, and the architecture

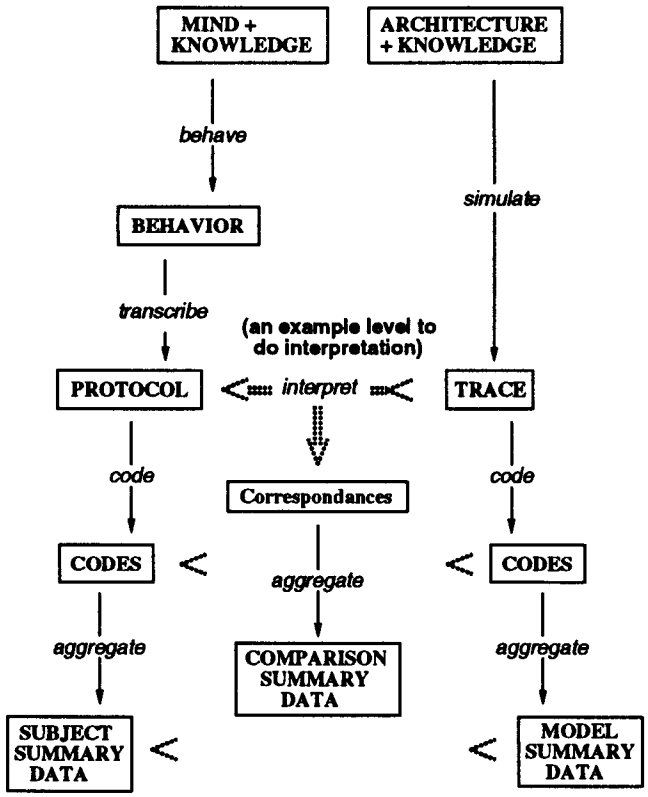

Figure 2-2: Information streams of subject and model used in testing process theories.

represents the fixed structure between tasks. Therefore, the model being tested consists of task knowledge and perhaps some general knowledge, within an architecture to apply it. Creating the model can be viewed as a type of programming task, of putting knowledge (programs) into the architecture (interpreter), except that testing the models is more difficult than testing programs because it is based on conforming to various non-obvious and indirect constraints, such as learning rates. If we choose the right architecture, we should be able to handle all situations. Modeling a situation then consists of specifying what knowledge is used in that situation.

The human information streams. The mapping between *mind+knowledge* and *architecture+knowledge* is not straightforward. We cannot open up a mind and examine its contents, we are only allowed to observe it *behave*. *Protocol* data are transcriptions of behavior that retains their sequential nature — data that remain ordered with and can be understood only with respect to other temporally contiguous actions. The data can include all types of human behavior: verbal utterances, key presses, and mouse movements. This data inherently occurs in time, and can have time stamps associated with it. This time stamp must be applied when it is *transcribed*, that is, recorded from the environment into a format that can be processed.

The sequential nature of the data can also be discarded. This results in non-sequential data that can be more easily aggregated and analyzed because it can be directly aggregated. Much of classical experimental psychology uses only behaviors' time stamps as non-sequential data, and while it is a valid subset, it is one with less information.

As a small example of how much information is lost when the time stamp is discarded, consider how much information is recorded when 64 subject actions are recorded, with these actions coming from 64 equally likely categories, and measured (time stamped) with 100 ms accuracy over a range 0 and 100 s. It takes 22 bits to record each measurement (6 for sequential position, 10 for reaction time, 6 for category). Discarding the sequential position of the measurement removes 6 of the 22 bits necessary to record each action.

Any transcription process will introduce noise and remove information. These losses may be small and systematic (e.g., most button presses can be recorded accurately to within 1 ms). The losses can also be large and disruptive. In particular, verbal protocols can lose information where the speaker did not speak distinctly, and noise (in the sense of erroneous information) can be added by misunderstandings of the transcriber. Information is also lost from the original protocol because such things as pauses, inflections, and environmental context are often not transcribed. Anything not transcribed is gone.

That is not to say that the protocol should not be transcribed, *coded* to create local summaries, or *aggregated* to create global summary statistics. In its original form the data may be unwieldy and the signal to noise ratio may be large. By condensing it one often ends up much better off; although less information remains further down the information stream, it is in a more understandable form.

The coding process can also be influenced by the level of the model and its stage of development. Models on the knowledge level (Newell, 1982) predict the knowledge applied. Lower level models may predict symbolic actions. The interpretation and comparison of the subject's actions with the model's would vary in each case. In each case the subject's behavior must be interpreted with respect to the model's predictions. Initially, when a model is being built, the codes may be derived from the data. When a model is being tested the categories are taken strictly from the model. Subject actions that cannot be interpreted this way are indicators where the model could be improved.

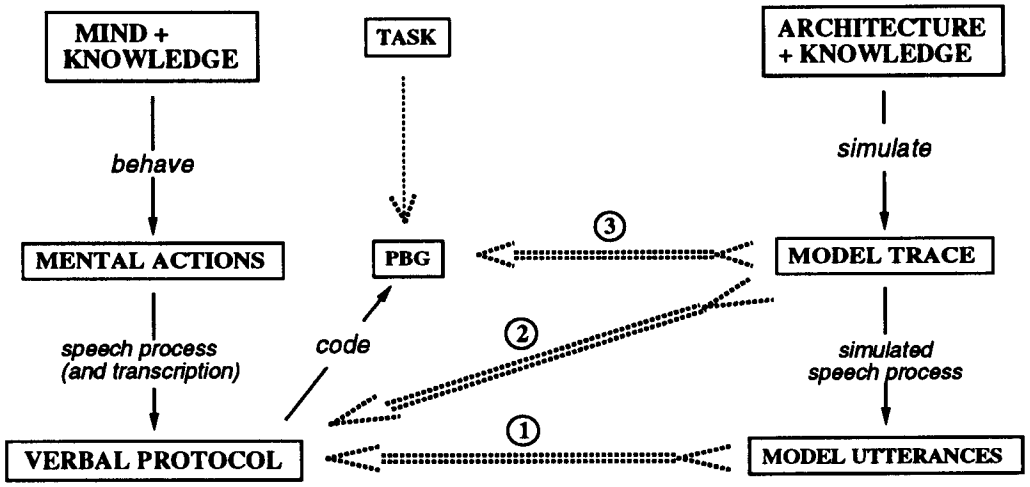

Figure 2-3: Schematic of possible levels of comparison between model and data. (Numbers are referred to in the text.)

The simulation information stream. Running the simulation *architecture+knowledge* (on the top left of Figure 2-2) produces the simulation's *behavior*. While this process should be completely accurate, the model traces never describe all the model's internal functions. The model builder often has a choice of how much information to make available in the trace. Like the *mind* information stream, the analyst

may further *code* and *aggregate* the information for subsequent analyses.

Comparing the information streams. After subject data have been collected and summarized to the desired level, and the simulation has been created based on task analysis and/or inspection of the behavior, the two information streams are ready to be compared. This will be done not only to indicate how similar they are, but also to provide information on how to improve the model. Figure 2-2 shows that there are several possible levels of comparison between the information streams as indicated by the arrows pointing in. A comparison consists of *interpreting* the protocol data with respect to the predictions of the model, to produce *correspondences*. Finally, these correspondences that can themselves be *aggregated* to produce *summary statistics* of the comparison, or displays indicating the major features of the model and how its predictions interpret the data. Although they will have a different form, the correspondences and summary statistics here serve the same role as residuals in linear regression, indicating the quality of fit locally and the direction of actions that must be taken to improve it.

The single most information-intensive approach is to compare the two information streams at the level of the *protocol* and *trace*. If the information in each information stream is coded before comparison, there is less information compared, but the comparisons may require less effort. If the model does not completely predict all the subject's actions on the protocol level, understanding the regularities that it does not match and where it needs to be improved can only come from comparing aggregate measures.

Some notes about verbal reports as data. For the most direct theoretical support, equivalent model and data constructs should be matched. If the process theories we are testing are about linguistic production, then verbal protocols are data on the most fundamental level. However, process models are usually not about linguistic production, so the verbal protocols must be interpreted in some way.

Figure 2-3 diagrams the relationship between verbal utterances and the process model's trace. If we are strictly to compare only equivalent items, we must compare the subject's overt task actions with the model's task actions, and the subject's verbal utterances with utterances generated by the model. This strict interpretation is depicted by the comparison arrow between verbal protocol box and the model utterances box (number 1). This approach has rarely been tried, but in a singular example Ohlsson (1980) showed that a simple model of utterance produced reasonable prose in a limited domain.

Fortunately, we do not have to add a speech process to our models. There is a theory of verbal protocol production (Ericsson & Simon, 1984) that under most conditions allows us to directly interpret the subject's verbal utterances as representing a subset of their mental representation of state information or operator application. This allows us to use the more direct comparison drawn between the verbal protocol box, and the model trace box (number 2), representing the comparison between the model's predictions of what could have been said, and what was said.

If the verbal utterances are difficult to decipher, we may choose to explicitly apply this theory of verbal production and create a coded representation of the subject's mental state (number 3) as a problem behavior graph, but we will take the approach in this work of using direct comparison wherever possible. It is simpler and subjects the data to one less transformation. Similarly, task actions are assumed to follow directly from their mental representation. The implementation of motor outputs is a separate process, but by definition cognitive models are not models of motor output. The effects of motor processing are usually kept small and ignored.

Predictions of actions should be matched by procedural data, and predictions of declarative mental structures should be matched by data representing declarative mental facts. Most verbal (and non-verbal) protocols in cognitive science are procedural data, a trace of an ongoing procedural task, which should be compared to the trace of a process model. When analyzing a protocol for declarative information, for example, content analysis (Stone, Dunphy, Smith, & Ogilvie with associates, 1966; Carley, 1988), matching verbal descriptions of declarative information to static declarative relationships is just the right thing to do.

We can now note the distinction between overt and verbal behavior as support for a cognitive model. Different aspects of the model match different data streams, that of verbal goals and states, and those of overt, motor actions. Both are support for the model, showing that it did predict mental or overt subject actions. The biggest difference is that the verbal utterances include an additional layer of theory specifying how they match, that of verbal protocol production.

Some analyses, typically those developing expert systems (Brueker & Wielinga, 1989; Shadbolt & Wielinga, 1990), compare procedural data to their static model. In this case, they are comparing disparate object, verbal utterances generated while performing the task to long-term knowledge structures or the static elements of a process model. Presumably this works well enough because they are using this to develop a model, which will not be tested on a fine grain level, and the rules are correct enough that they would fire in such a situation. Given a complicated enough model, the emergent properties of the model will prevent matching the procedural data to a description of a procedural model. For example, new operators could be created from existing knowledge or from new knowledge learned from the environment.

This view of protocols as data to test models suggests that the analyst is not just trying to "assign" a trace element to a subject segment, with that accounting as the sole result. It is not. The assignment of segments is a way to test a model, finding support for its components in the protocol. This approach equally includes the desire to know where the predictions fail to match the protocol so that the model may be improved.

2.2 Review of creating and testing models with protocol data

The framework shown in Figure 2-2 supports the comparison of research approaches based on which information streams they use, the processes they use to transform data, and any tools used to automatize the transformations or aggregations. Consider as a straightforward initial example under this framework, experimental psychology. It generally lives only on the left hand side information stream of human data, and as mentioned above, on a subset of that data even. Reaction choices and times are taken, often automatically coded by computer-based apparatus into categories and aggregated into means and other numeric measures by common statistical software.

As a more complete example, consider Qin and Simon's (1990) work examining the process of scientific discovery. Subjects (mind+knowledge) talked aloud (producing a protocol) while they attempted to find Kepler's laws of planetary motion based on Kepler's data. The subject's utterances were recorded on audio tape and transcribed into text (protocol). The subject data was coded and aggregated by hand into the functions examined, showing that the subjects' work appeared to be done as search in problem spaces (summary data). On the model side, they gave Bacon, a scientific discovery system, the same task. They took a trace of its operations while it solved the problem. The trace was coded and summarized by hand. The aggregations summarized Bacon's behavior as search in problem spaces (summary data). Qin and Simon then informally compared the summary of the subjects' behavior with that of Bacon, and were able to conclude that the mechanisms in Bacon are sufficient to account for the subjects' ability to perform the task, and the style in which they did it. (interpret and code). Their comparison (as briefly explained in this simplification) is easy to follow partially because it used only summary data.

2.2.1 Exploratory analysis leading to process models

Figure 2-4 depicts several of the levels of theory that are built and tested with protocol data. Levels near the top represent exploratory analyses necessary for creating models that can later make more concrete predictions of behavior. Analysis of the data creates the model. As the model becomes more predictive (middle line), it is derived less from the data at hand and attempts to interpret the data. When a model makes predictions of actions sequences (bottom line), it becomes a process model. It is no longer directly influenced by the data during analysis. Its predictions are first compared to the data, and the areas where they mismatch may suggest where to modify the model.

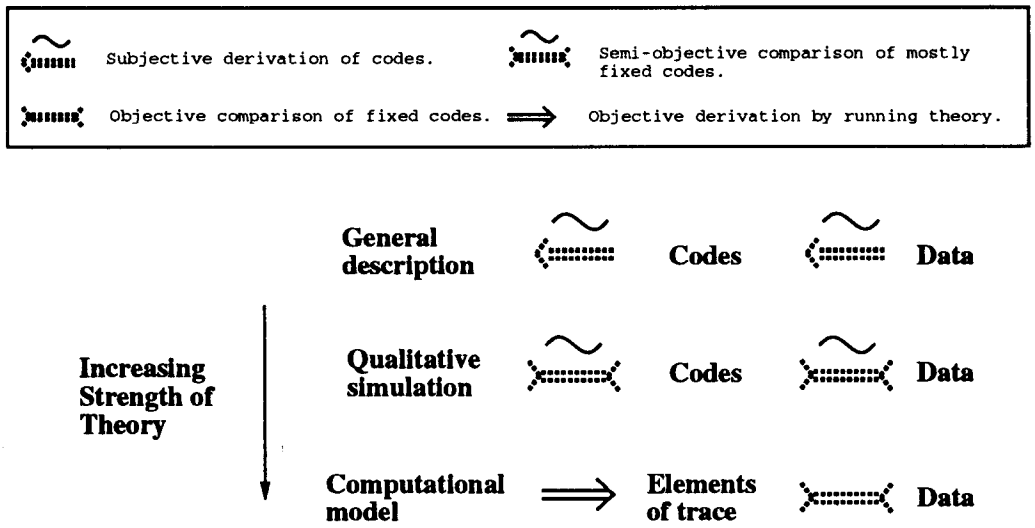

Figure 2-4: How theory level influences comparison with data.

It is worth noting how process models are developed, for many of the same subtasks will be used in refining them after testing. The first line in Figure 2-4 refers to coding that starts out without a model of the subject's processing or knowledge and leads to at least a mental simulation of the subject's structures. The coding process is data driven and is done to derive the underlying actions and the process generating them. This work can be characterized by categories that are general and that may change during the analysis. The first analyses of the data provide only aggregate measures of the human behavior. This exploratory coding works primarily with the human information stream. There is little formal or direct comparison with a model. The aggregate measures produced by the analyses create constraints and hints for creating a model that can be mentally simulated.

Table 2-1 shows several examples of each types of analyses, including two experimental psychology studies for comparison. From left to right, its columns are Citation, a citation for the study; Domain, the task domain studied; N, the number of subjects used in the analysis; Data Points, the total number of data points as defined by the experimenter or protocol segments, analyzed or compared to a model; Total time, the total subject time in seconds included in the analysis; Data types, the data types taken, in the first subcolumn S is sequential data where each action is ordered in time, N is non-sequential data where the subject makes a single response, like an answer to an arithmetic problem, O is overt task actions, I is eye data, and V is verbal data; Codes, the type of codes derived from the data for the analysis, as in Figure 2-2; How, how the codes were assigned to the data, by hand, semi-automatically with a tool, or automatically by a tool; Model compared, how the model was compared with the data or what analyses were performed; Model, the type of model tested with the data.

Examining the data in this way is necessary when a model is not available or not yet developed, when a complete understanding is not necessary, or when time is not available to derive it. For example, fast user testing of computer interfaces (Kennedy, 1989) by categorizing user's actions into errors and correct keystrokes does not require a complete model to indicate where they may need improvement. This style of work particularly exists in areas that are trying to create initial models (such as computer supported cooperative work; Olson, Olson, Storrosten, & Carter, 1992) and knowledge rich areas that are still difficult to model (for example, computer programming, Fisher, 1987, and process control, Sanderson, Verhage, & Fuld, 1989). Taken as a group, these types of studies (shown in Figure 2-1) require relatively larger data sets than are used for testing process models, and verbal data are always used, and sometimes overt task actions are also collected.

Citation	Domain	N	Data Points	Total Time	Data Types	Codes	How Coded	Model compared or Analysis Performed	Model
Typical experimental psychology studies: (examples)									
Reder & Ritter 1992 - I	Arithmetic	34	40k	146 ks	NO..	RT	auto	regression	Log&LinReg
Woltz 1989	Skill acquisit.	701	1.3M	9 Ms	NO..	RT	auto	regression	LinReg
Exploratory coding of verbal protocol: (examples)									
Fisher 1987	Programming	3	na	18 ks	SO.V	Operator	semi	cycle detection	op names
Bree 1968	Reading	12	~4k	21 ks	S..V	Strategy	hand	auto PBG f/ codes	qualitative
Olson, et al. 1992	Design w/ CSCW	6	na	32 ks	SO.V	Action	hand	counts	none
Sanderson et al. 1989	Process control	12	~2.9k	108 ks	SO.V	State	semi	transitions	state types
Process-based trace comparisons (examples):									
Dillard et al. 1982	Accounting	3	210	na	S..V	KnowLev	semi	match to codes	fixed manual
Wagner & Scurrah 1971	Chess	2	271	7 ks	S..V	State	hand	rule match to state	gen'l rules
Bhasker & Simon 1977	Chemical Eng.	1	280	na	SO.V	Operator	semi	assignment to model	fixed manual
Gascon 1976	Seriation	33	1153	na	SO..	Operator	hand	match to codes	structural
Development and general comparisons with procedural model (examples):									
Anzai & Simon 1979	Tower of Hanoi	1	224	5 ks	SO.V	State	hand	informal comparison	simulation
Karat 1982	Tower of Hanoi	192	10k	~6 ks	SO..	State	auto	linreg on time/rule	simulation
Ohlsson 1980 - II	Lin. syllogism	10	1.5k	17 ks	C..V	PBG	hand	informal	qualitative
Qin & Simon 1990	Sci. Discovery	6	na	32 ks	SO.V	State	hand	aggregates comparison	simulation
Simon & Reed 1976	Mis'ry/Can'bls	100	2500	na	S..V	State	hand	relative state rates	Markov
Atwood & Polson 1976	Water jugs	229	13.5k	na	SO..	State	auto	relative state rates	sim & Markov
Hegarty 1988	Diagram use	40	4.8k	36 ks	S.I.	Vis. area	auto	informal comparison	simulation
Klahr & Dunbar 1988	Sci. Discovery	30	na	~48 ks	SO.V	State	hand	informal comparison	description
Newsted 1971	Concept Ident.	2	~1.2k	144 ks	SO.V	KnowLev	hand	aggregate	simulation
Aggregate model time stats vs. Subject states (example)									
John 1988	Typing	29	regularities			Aggregates	hand	qual & quantitative	process
Full trace based comparisons, non-sequential response data (example):									
Thibadeau et al. 1982	Reading	14	~27k	7 ks	N.I.	RT	auto	regression on time	simulation
Category coding of declarative verbal statements (examples):									
Carley 1988	Relationships	16	na	115 ks	N..V	KnowLev	semi	insert into net	sem nets
Stone, et al. 1966	(varied)	>100	>100k	360 ks	N..V	Concept	auto	content analysis	none

In the data type column: N indicates non-sequential data. I indicates eye tracking protocols.
S indicates sequential data. V indicates verbal protocols.
O indicates overt task actions protocols.

Table 2-1: Examples of protocol datasets, their sizes and ways they can be used to build and test process models, with example experimental studies for comparison.

As a final analysis, exploratory coding has several drawbacks compared to one done with a model in hand. By creating the codes from the data, it decreases the degrees of freedom in the analysis (Ericsson & Simon, 1984). Without a functional model in hand, the codes may be unequal sizes, and aren't defined operationally. As new ideas are explored the comparison between predictions and observations can end up being informal. Unless rigorous coding schemes are put in place for the coders, the codes may be difficult to keep in focus as they will change as more data are coded, and the model is further developed.

As the analyst forms a mental model of how the subject performs the task, the coding task changes to semi-theoretical coding (Figure 2-4, line 2). The derivation will be incomplete, but will be better motivated than those with no model in mind at all. Analysts with a mental model in mind may be informally comparing the subject's action sequence with the action sequence or its components that the mental model might produce. The codes will have more meaning, corresponding more directly to mental operations or mental state information, but are not yet completely formalized. Typically the aggregate measures taken are the number of codes in each category. Detection of behavior cycles through such measures as sequential lag analysis (Gottman & Roy, 1990) becomes more important because they are used to suggest action orderings for the model. As a step towards process models, users may also create problem behavior graphs of subjects (Bree, 1968).

When the process model is not yet implemented as a simulation, but its actions are fairly well defined, the subjects sequential behavior can still be examined by comparison with something other than a trace. Bhaskar and his colleagues (1978; Bhaskar & Simon, 1977; Dillard, Bhaskar, & Stephens, 1982) compare actions in the subject's sequential behavior to the steps in a block diagram of the process. The coded protocol can also be examined by a pattern matching program to find strategies based on their signature patterns (Gascon76, 1976), or some of the rules for behavior can be tested by hand against the data (Wagner & Scurrah, 1971). As shown in Table 2-1, these studies typically use less data than more exploratory ones.

2.2.2 General testing of process models

The last line of Figure 2-3 represents testing the sequential predictions of a process model with data. Once a process model has been created, it can be run on the studied task, generating a trace of its implicit and overt behaviors. This is the model information stream in Figure 2-2. The model's actions are no longer being derived from the data, but the model is attempting to predict the data. This is a more powerful result than the transition probabilities that sequential lag analysis provides.

There are many degrees of freedom in these models, which correspondingly require large amounts data to test them. Each production (or subprocedure specifier) is like a parameter in a regression, in that it is included to explain or account for some of the data. Therefore, proportional to the degrees of freedom in the model, large amounts of data are needed to create and test process models. This data has been found in verbal and non-verbal protocols, and sometimes from aggregate measures taken from other studies.

Process models are more difficult to build and manipulate than verbal theories because they are more detailed and make more direct predictions of the subject's action sequences. We can also take additional measures from them as they perform the task. These measures have included the time to do the task (Carpenter, Just & Shell, 1990; Dansereau, 1969) and information attended to in the environment (Hegarty, 1988).

Most often these models have been tested against the data through aggregate measures. In the terms of the diagram depicted in Figure 2-2, the model is run and its behavior is summarized into aggregate measures, which are then compared with the aggregate measures taken from several subjects. These aggregate measures have included state transitions (Atwood & Poulson, 1976; Simon & Reed, 1976), time to make state transitions or apply operators as indicated by overt task actions (John, 1988; Karat, 1968), and eye movements (Hegarty, 1988; Just & Carpenter, 1980). More often just general regularities that subjects exhibit are compared with general descriptions of the model's behavior, such

as strategy preferences (Hegarty, 1988; Klahr & Dunbar, 1988; Qin & Simon, 1990; John, 1988). These studies use more data than those that start to test process-like models, partially because the data set is also used to create the model.

2.2.3 Trace based protocol analysis

The third line of Figure 2-4 depicts directly comparing the process model's trace with the subject's actions and verbal utterances without aggregation of either information stream. This is called model-trace based protocol analysis by Ohlsson (1990), and trace based protocol analysis in this thesis.

When subject actions are assigned to match a trace segment of the model, the codes assigned have a deeper meaning; segments of data are not being grouped into general categories, but are being assigned as supporting a specific, formal prediction from an IP model about the type and order of the action sequences used to perform a task. The organization of the model determines which actions and codes are similar and can be aggregated.

Process models have been primarily tested on this level with three types of data: verbal protocols, overt task action protocols, and eye movement protocols. Verbal protocols were the first to be developed as a way to test process theories. As specifications of working memory and operator applications (Ericsson & Simon, 1984) they provide the most detailed data points to match. The top section of Table 2-2 presents a list of studies that have tested process models with verbal protocols. It is nearly a complete list, and there are probably no more than ten more models tested with verbal protocols. In sharp comparison, consider the position of another, much simpler modeling technique, multivariate analysis. In 1980 it was estimated that there was over 10,000 papers in the literature (Bentler, 1980).

As a group these studies are fairly homogeneous. They nearly all deal with problem solving on a puzzle type task and studied a small number of subjects. There are four studies that then stand out. Three do so because of their size. The original Newell & Simon (1972) work, Ohlsson's (1980) second study, and Larkin, McDermott, Simon, & Simon (1980), all used more than one subject, and examined relatively large amounts of data. Peck & John's (1992) work stands out because it is not of problem solving, but of performing what is claimed to be routine behavior, that of using an on-line help system. In all cases, the number of verbal segments that have been used to directly test the sequential predictions of a process model are relatively small.

Overt task actions have been used most often to test process models, so often that it is not possible or interesting to enumerate all of them. They have been taken for a variety of tasks and a variety of methods. The most common types that have been used are mouse and other computer input devices (see Table 2-2 for examples). The protocols are often directly taken by computer, but don't have to be. Transcribed or actual pen actions have been used to model subtraction. Young and O'Shea (1981) used marks generated on paper as byproducts of the task of subtraction to test their theory of subtraction bugs, and VanLehn and Ball (1987) used ones transcribed by an electric pen. While eye movements are fairly routinely used to develop informal models (Hansen, 1991) and process models (Reader: Just & Carpenter, 1980), they are probably the least used to test process theories directly, and are rarely as the only data source. Table 2-1 indicates that when the overt task responses can be treated non-sequentially, relatively large amounts of data can be compared with the model's predictions (e.g., Thibadeau Just, & Carpenter, 1982).

Some models have been tested with more than one type of data, and doing this appears to be useful. Young (1972) used primarily task actions (moving blocks) along with some eye movements to build and test a model of children solving seriation tasks. Peck & John (1992) used verbal utterances and task actions (mouse movements and clicks) to build and test a model of a user searching for help in an on-line browser. Newell and Simon (1972, pp. 310-327) report on Winikoff's work (Winikoff, 1967) using eye movements as an adjunct to verbal protocols.

Citation	Domain	N	Data Points	Total Time	Data Types	Coding	How Coded	Model compared or Analysis Performed	Model
Full trace based comparisons, continuous data: (believed complete)									
VanLehn 1989 - I	Miss. & Cann. 5	1	9	na	S..V	State	hand	trace	simulation
Ohllson 1980 - I	Lin. syllogism	1	56	220 s	S..V	PBG	hand	trace	simulation
Greeno 1978	Geometry	6	72	na	SO.V	KnowLev	hand	trace	simulation
Koedinger 1990	Geometry	5	98	na	SO.V	Operator	hand	trace	simulation
Luger 1981	Physics	5	102	na	SO.V	KnowLev	hand	general trace	simulation
Feldman 1959	Binary choice	1	200	~1 ks	S..V	Trace	hand	trace	simulation
VanLehn 1991	Tower of Hanoi	1	224	5 ks	SO.V	State	hand	trace	simulation
Newell 1990 p364+	Cryptarithmetic	1	238	200 s	S..V	PBG	hand	trace-search control	simulation
Newell & Simon 1972	Chess	1	242	na	S..V	PBG	hand	rule trace	hand sim.
Larkin et al. 1980	Physics	21	254	~8 ks	SO.V	KnowLev	hand	trace	simulation
Johnson, et al. 1981	Heart diagnosis	12	264	na	S..V	State	hand	state trace	simulation
Ohllson 1980 - II	Lin. syllogism	5	~400	~2 ks	S..V	PBG	hand	trace	simulation
Baylor 1971	Visual puzzle	1	423	727 s	S..V	PBG	hand	production trace	simulation
Peck & John 1992	Help menus	1	624	~600 s	SO.V	Operator	hand	operator trace	simulation
Moran 1973	Visual imaging	1	656	1.2 ks	S..V	State/Op	hand	trace	simulation
Newell & Simon 1972	Cryptarithmetic	1	1.9k	810 s	S.IV	PBG	hand	rule trace	simulation
Newell & Simon 1972	Cryptarithmetic	5	2.0k	10 ks	S..V	PBG	hand	rule trace	hand sim.
Newell & Simon 1972	Logic	10	3.7k	na	S..V	PBG	hand	rule trace	simulation
Jones & VanLehn 1992	Physics	9	4.3k	na	S..V	KnowLev	hand	trace	simulation
Full trace comparison with non-verbal: (examples)									
Newell & Simon 1972	Chess	1	16	na	SO..	Operator	hand	overt actions	simulation
John et al. 1991	Nintendo	1	73	27 s	SO..	Operator	hand	trace	simulation
Ruiz & Newell 1989	Tower of Hanoi	3	~96	na	SO..	Operator	hand	trace	simulation
Young 1973	Seriation	11	~570	~760 s	SOI.	Op & prod	hand	trace	simulation
Young & O'Shea 1981	Subtraction	33	>2k	na	SO..	Prod	hand	trace	simulation
VanLehn & Ball 1987	Subtraction	26	~3k	na	SO..	Operator	na	operator constraints	simulation
Koedinger 1990	Geometry	30	na	864 ks	SO..	Operator	auto	trace	simulation
Anderson and group	Tutoring	>300	>500k	>10 Ms	SO..	Prod	auto	trace + reset	simulation

In the data type column: N indicates non-sequential data. I indicates eye tracking protocols.
S indicates sequential data. V indicates verbal protocols.
O indicates overt task actions protocols.

Table 2-2: Summary of previous uses of protocol data to test process models.

2.2.4 Summary of important data features

With the studies reported in Tables 2-1 and 2-2 spread before us, we can examine what general patterns characterize the testing of process models, particularly testing with protocol data.

1. Verbal reports are not yet treated as data. The amount of protocol data used to test the model is often not directly reported, as indicated by the relatively large number of not available (na) or approximated (indicated with a ~) measures in Tables 2-1 and 2-2. Researchers appear to routinely obtain more protocol data then they use to test the model. Sometimes this is explained by subjects performing too randomly to code, but often it appears that they lacked the resources to take advantage of the data. In either case the fail to report its amount and disposition as carefully as reaction time data are treated.

2. Testing process models with verbal protocol data appears to take a lot of effort. The testing is seen as tedious and dangerously boring, offering plenty of opportunities for mistakes (Ericsson & Simon, 1984, p. 271). Generating and testing process based theories by hand takes a lot of time and is tedious. The total analysis time to subject behavior time may be greater as 3600:1 if you base the measures on reported lengths of studies (Ohlsson, 1980). The testing of a model may account for perhaps 1/4 to 1/2 of the total time to develop a process model (Ohlsson, 1992). The number of categories that are used to code that data is the number of operators or trace elements. Each of the trace elements will require a rule for coding it. If the analyst is someone experienced with simulations they may get by with fewer rules. Later steps of modifying the model require someone who is both experienced with simulations so that they can create and modify the process model, familiar with psychology so that they can understand additional behavior constraints, and patient enough to do the alignment and realignment to get the best fit between data and model.

3. When process models are tested they are not tested with a lot of data. The difficulty of testing is also supported by how often process theories are created versus how often their sequential predictions are tested. While process models are now routinely developed, they are not often tested. Testing may become semi-routine within a single study (e.g., Peck & John, 1992), but there appears to be no routine use of protocol data to test process theories. Only four of the 27 process models designed or implemented as computer programs that appeared in Cognitive Science between 1980 and 1986 compared the model's behavior with a subject's action sequence, although several more compared the aggregate performance of subjects and the model (Kaplan, 1987). If model creation was the only significant cost to developing process models, one would also expect the developers to amortize this cost by testing additional subjects, or using more data per subject. As shown in Figure 2-2, the number of subjects and measures used to test these theories is small. Often papers compare a model with the actions of just 1 subject. Sometimes models are compared with up to 5 subjects, but rarely, if ever, are the action sequences of 30 to 100 subjects compared with the model's actions. In addition to a small number of subjects, the total amount of subject actions included in the comparison is often quite low. It is not particularly uncommon to see less than 1000 s (16 minutes) of subject data, and the largest amount of subject data compared with a single model's actions is on the order of several thousand seconds. Sometimes process models are compared with large numbers of subjects over hour-long time periods, but then they are only tested against aggregate measures. Testing process models with the amount of data in the example experimental psychology subject numbers and contact times per study shown in Table 2-1 (20 subjects by 1 hour) is out of reach. Large experimental studies with 700 subjects for 3.5 hours are inconceivable. This comparison process appears so difficult to some, that only general requirements are referenced (see Kaplan (1987) for further citations), or anecdotal evidence is used without reference to subjects at all (Schank, 1982).

4. Testing process models with verbal protocol is limited to a few scientific centers. Sociologically, the methodology of trace based verbal protocol analysis has not become widespread. All of the researchers using verbal protocol to test process models (with a few exceptions) started using it while students or associates of Newell or Simon. If we include researchers that use non-verbal protocols we find that they are more widely dispersed, but still centered around Carnegie-Mellon and the University of Colorado at Boulder, and that few investigators have done more than one such study.

5. Sequential data are more difficult to use. Based on the number of task actions per task, subjects' actions can be categorized into two groups, sequential and non-sequential. Sequential data represent ongoing activity, are ordered in time, and are contingent on previous actions. For example, the actions taken to solve a cryptarithmetic problem represent an ongoing process, and the verbal protocol generated must be understood with respect to what has gone before it. Solitary data are subject actions that represent unitary responses to the task. Simple responses to subtraction problems are an example. There is a continuum. The number of segments in a protocol can be one, like in subtraction, or number four or so, as in solving physics textbook problems (Larkin, et al., 1980; Luger, 1981), or number in the hundreds, as in cryptarithmetic. As long as there is more than one of them, they are sequential. If learning is included (which is rare), then even the single responses are no longer non-sequential. Longer groups of sequential data are used less often, appear to be more difficult to model, and should be more difficult than solitary data because the model must fit for longer periods of time.

6. Discrete data are easy to interpret and helps interpret other data. Discrete data are taken to mean data points that are individually distinct, unconnected elements, taking on a finite (or countably infinite) number of values. The transformation from data to codes in Figure 2-2 is straightforward because the data already represent actions on the task level. If the data are fully discrete, then the testing can be more easily automated, and the full power of visual display and human mediated analysis is not needed. Having directly interpretable data included in the protocol helps to tie down with respect to the model trace less directly codable data, such as verbal protocols. Mouse clicks are presumably discrete; mouse movements in x,y coordinates are not. Eye movements once interpreted as regions of a diagram or as words in a text are discrete (before that they are not). Verbal protocol is not discrete, it can take on many values that map to the same representation, all of which require interpretation. Producing discrete data requires an interface (usually on a computer) that dictates the subject's interactions. This limits the types of tasks it can be collect from, and the actions subjects can use to perform the task.

7. The level of the model affects the amount of data required. The grain-size of the process model and its stage of development affects the amount of data that can be used and the size of the task that can be approached. Initial development of a model usually requires more information than is used to test it. Theories done on a larger grain-size will have a process trace that matches larger lengths of subjects protocols than more fine grained analyses. Models of actions on the knowledge level (Newell, 1982), such as Able (Larkin, et al., 1980), will match large portions of data (perhaps 20 seconds), and models of actions on the keystroke level, such as Browser-Soar (Peck & John, 1992), will match small portions of data (taking seconds or several hundred milliseconds). There are classical tradeoffs here. Higher level models account for more data. More detailed models use less data, but account for it in a more detailed way. The success of closer encoding to a model comes at the price of more comparisons for a given stretch of time. The very direct lesson then, is to model only the appropriate results, not any unnecessary details.

2.3 Tools related to process model testing

There are numerous processes and capabilities required for building and testing process models. The tools to do this can be grouped by primary capability. Examining them will highlight useful features for testing, understanding, and manipulating process theories. There are no tools for routinely testing process models, but there are often tools to perform many of the subtasks. Process theories are often started from simply examining protocol data, and that is the first tool type we examine. A shorter, and slightly different view of available tools is available in Sanderson, James, Watanabe, & Holden (1990).

2.3.1 Tools for building models from protocols

2.3.1.1 Declarative knowledge coding tools

The simplest way to code a verbal protocols or texts is to note the concepts or declarative knowledge that make up each segment. The sequential nature of the protocol is ignored, and the coding is done in a straightforward manner. While the tools for coding declarative knowledge differ fundamentally in function, they hold a direct lesson for process model testing tools. They show how successful semi-automatic and automatic tools can be. The first of these was the General Inquirer (Stone, et al., 1966). It routinely took in a 100s of thousands of data points (as words from texts), and automatically categorized them into semantic types based on a database of words it understood. It was widely and apparently easily used to do content analysis. More recently, Carley (1988) has built a less automatic, but more theory guided set of tools to code knowledge in texts into semantic networks. Segments are presented automatically to be coded, and there are tools to check to make sure the networks are built correctly and consistently. The major analyses she supports are how much knowledge is shared between subjects or within a subject across time. These tools are related to all simple text processing tools, such as automatic indexers (Waltz, 1987).

2.3.1.2 Exploratory protocol analysis tools

Tools that support the levels of exploratory analysis and qualitative simulation (lines 1 and 2 in Figure 2-4) share many of the same features and are best discussed together. The success of these simple tools establishes a baseline of manipulation requirements and shows that computer assisted coding and tabular displays of data are useful.

Simple coding tools. Simple coding tools are defined by the limit of the aggregate measures that they provide. The only aggregation method that they support is the ability to sum the number of times each code has been used. At a minimum, simple coding tools assist the coding process by (a) presenting menus of codes, (b) support treating the text as a set of segments so that the analyst can manipulate and move between segments directly, and (c) support aggregating the codes. In addition to these direct needs, enough editing of strings goes on that a full word processor ends up being generally required. There are numerous tools running on a variety of platforms that provide no more than this (e.g., VPA: Brown, 1986, Lueke, Pagerey & Brown, 1987; PAP: Poltrock & Nasr, 1989; Cref: Pitman, 1985; and others presented in Fielding & Lee, 1991). This level of tool may also assist by automatically presenting segments to the analyst to code, such as MPAS (Erikson & Simon, 1984) was explicitly designed to do.

Recently analysts have started coding the protocol on the video tape directly to represent the protocol (Kennedy, 1989; Mackay, 1989; SigChi, 1989). Video is more compelling data representation because it is more complete, but it is more unwieldy. When the protocol exists on video, more specialized tools are necessary and available. Sometimes the video based tools are just simple coding tools, and some of them include additional features making them exploratory data analysis tools.

General purpose tools have also been appropriated to the task of coding. Analysts have used spreadsheets (Excel: Peck & John, 1992; Lotus 1-2-3: Cohen, Payne, & Pastore, 1991), databases (Filemaker, Mac-Allegro and DataDesk: Larkin, personal communication, July 1992), and even multi-

column text editors (Prep: Neuwirth, Kaufer, Chandhok, & Morris, 1990). These tools offer a level of polish that the research tools don't, but they fail to be integrated into other programs, or be extendable to directly interact with specialized analyses.

Table 2-3: Types of protocol analysis tools and their features.

		General Features			Trace Alignment		Model Incorporated			
TOOL TYPE	CODING AIDS	COUNT CODES	TABULAR DISPL	MACROS /EXTEND	PLAIN	AUTO-MATIC	INTE-GRATED	PROCESS BASED	FLEX-IBLE	AUTO-MATIC
DECLARATIVE (eg, Carley 1988)	+/-	X	+/-	.	.	.	+/-	.	X	.
SIMPLE CODING (eg, VPA: Brown 1986)	+/-	X	.	.	.	.	.	.	.	.
EXPLORATORY (eg, SHAPA: James et al. 1990)	Semi	X	.	.	.	.	.	.	.	.
GENERAL PURPOSE (eg, Excel)	Semi	X	X	X	X	a	.	.	.	.
KNOWLEDGE ACQUISITION (eg, Keats: Motta et al. 1988)	+/-	.	.	X	.	.	+/-	X	X	+/-
INDUCTION TOOLS (eg, Cirrus: VanLehn & Garlick 1987)	Hand	.	.	X	.	X	X	X	X	X
PROCESS BASED (SAPA: Bhaskar & Simon 1977)	Semi	X	.	.	X	.	X	.	.	.
GENERAL PROCESS BASED (Pas-II: Waterman & Newell 1973)	Auto	.	.	X	X	.	X	X	X	.
INTELLIGENT TUTORING SYSTEMS (eg, Angle: Koedinger 1990)	Auto	X	.	.	.	X'	X	X	.	+/-
GENERAL TRACE-BASED (Trace&Transcription, John 1990)	Semi	X	X	X	X	.	.	X	X	.
Soar/PA	Semi	X	X	X	X	X*	X	X	X	.

Legend:
- . Feature not provided.
- +/- Varies between tools in this category.
- X Feature provided.
- X' but not generally available to analyst.
- X* based on keyword matches only.
- Man Done manually.
- a Could be added.

ESDA tools. Exploratory sequential data analysis tools represent a slightly higher level of functionality (e.g., Paw: Fisher, 1991; Shapa: Sanderson, 1990). They tend to support more detailed coding schemes, more closely tied to theoretical constructs, and more sophisticated aggregating measures. They typically directly support a method for detecting loops in behavior, such as sequential lag analysis (Shapa, Paw), or maximum repeating pattern (Siochi & Hix, 1991). These tools are not useful for testing process models because (a) they don't include a process model, (b) they do not generally provide tabular displays, and (c) they lack an automatic interpretation and alignment routine and the facilities to easily incorporate one.

Childes-Clan (MacWhinney, 1991; MacWhinney & Snow, 1990) is the most sophisticated of these programs, perhaps because it was developed for working with children's utterances, which have a greater variety of styles of speaking than problem solving protocols. It is probably the most widely used, with approximately 250 users (MacWhinney, personal communication, October 1992). It includes a structured editor for semi-automatic coding, a language for creating hierarchical codes, the ability to perform numerous types of counts, and to compute context sensitive measures. It comes with several hundred megabytes of transcribed and annotated protocols.

Programs to summarize the data can also exist on their own. Bree (1968) wrote an analysis program that took in coded protocols, and created a type of problem behavior graph that highlighted the cycles in the subject's behavior.

2.3.2 Model testing tools

2.3.2.1 Strategy classification tools based on process models

Tools have been developed that test process models, but they do not directly compare the protocol with a process model trace. These include tools that set model parameters based on the protocol (ASPM: Polk, 1992), that compute the order of unit tasks (SAPA: Bhaskar & Simon, 1977) or knowledge applications (KO: Dillard et al., 1982) in the protocol, or that characterize the protocol as matching one out of a set of strategies (Gascon76, 1976).

Given a model that performs the task, a list of model parameters and how they vary its performance, and behavior to model, ASPM (Polk, 1992) will find the optimal setting of parameters. Its strength is that finds an optimal setting by searching efficiently over huge combinatorial spaces. Its drawback is that it requires a lot from the model and modeler. The model must be expressed in a set form, a complete specification of parameters is required, and a way to derive the models outputs automatically for each input must be provided. Each step in a process must be treated individually, testing a process model thus consists of numerous individual tests.

SAPA (Bhaskar & Simon, 1977) starts out with a fixed flow diagram model that predicts the subjects unit tasks, their possible orderings, and the decision points that subjects will use to solve a general class of problems. In Bhaskar & Simon (1977) the model is for solving thermodynamics problems. Analyzing a protocol consists of assigning each segment to a task. When segments match, the model moves ahead and proposes the next step to match the next segment. On mismatches, the analyst is expected to reset the model appropriately. No provision is made for modifying the model under test. Segments can be presented to the analyst automatically for coding. Accounting, thermodynamics and business policy problems have also been used with SAPA (Bhaskar, 1978), and Dillard et al. (1982) develop a similar system called KO that works for declarative knowledge in accounting.

Gascon (1976) in his thesis wrote a program in Conniver and Lisp to recognize a fixed set of strategies (ten in all) for a weight seriation task. Based on codes assigned by hand to the subject's actions, his system recognizes the global strategy and assigns the individual actions to particular steps of the strategy chosen.

2.3.2.2 Model tracing modules within intelligent tutoring systems

There are at least a few examples that show that the action sequences generated by process models can be compared automatically with human behavior. Some intelligent tutoring systems (particularly Anderson and his group's tutors (Anderson, Farrell, & Sauers, 1981; Anderson, Greeno, Kline, & Neves, 1981; Reiser, Anderson, & Farrell, 1985; Singley, 1987; Singley & Anderson, 1989; but also Sleeman, Hirsh, Ellery, & Kim, 1990, and VanLehn, 1983) provide these exceptions. Subjects as students now routinely work with Anderson's tutors, and have their actions compared with a process model of geometry proof construction (Koedinger & Anderson, 1990) or lisp programming (Anderson, Farrell, & Sauers, 1981). The process models used in the tutors come from previous non-automatic

analysis (Singley & Anderson, 1989), but sometimes they are able to make use of the tutor format to refine existing theories (Koedinger & Anderson, 1990).

Closer examination reveals that this extreme level of automation requires several constraints that are not always available or acceptable: (a) The domain is not open ended, there are fixed operations and responses. (b) All subject actions are readily interpretable because they must come from a fixed set available on menus, and no verbal protocols are taken. (c) The subject model is a well developed one, so that when the match is done automatically it will be a close fit that is easier to do. (d) The nature of the tutoring task and mature models allows the analyses to be done in a closed manner — when the subject's protocol mismatches the model, the subject as a student is reset and told to start over.

2.3.2.3 Tools for aligning the sequential predictions with data

A necessary step in analyzing the predictions of a process model is to bring the predictions into correspondence with the sequential actions of the subject. The model and subject have to be initially synchronized, and then after each time they lose step with each other. This discussion assumes that both sequences are in already in hand, and that the alignment does not have to be computed dynamically.

Two general systems have been created to test unrestricted process models by aligning their predictions to the data either semi-automatically (Pas-II) or by hand (Trace&Transcription). In both cases they do not directly incorporate the model being tested, but provide a framework for comparing the model's predictions with protocol data.

Aligning the model's predictions with the subject's actions and verbal utterances would be particularly nice to automate. The alignment has mostly been done by hand; performing this automatically would remove a tedious task for the analyst. Performing the comparison automatically will require that the process is fully specified, which will also make the prediction testing more reliable. No extant cognitive modeling system or protocol analysis systems supports this, although it has been done automatically on a small scale to explicitly test small models (Card, Moran, & Newell, 1983).

PAS-I and II. Pas-I (Waterman & Newell, 1971) was a system to automatically analyze verbal protocols on cryptarithmetic. Pas-II (Waterman, 1973; Waterman & Newell, 1973) was a generalized version of Pas-I. They provide a complete model of how to code verbal protocol into problem behavior graphs. Their strengths were: (a) They made the analysis process very objective and explicit. (b) They were complete and integrated. One could use them to code verbal protocols from transcript to problem behavior graph. (c) They were both flexible. Pas-I could be modified to deal with any cryptarithmetic protocol, and Pas-II could be modified to work with any domain.

Despite their strengths, they appear to have been applied only to their initial test data. Their weaknesses may be even more enlightening than their strengths:

- They appear to have been complicated and hard to use. Pas-II, the general system, required the user to represent all the knowledge for each of the analysis steps as productions. Facilities for creating, editing, and testing these productions do not appear to have been provided.
- The number of steps may have seemed daunting. There were 21 steps implemented out of 28 steps in the complete design. Each step required it own mini-production system for coding the protocol.
- They did not directly include the process model they were attempting to match. This means that the same information for interpreting the data with respect to the model had to be represented in several locations.
- The output of the analysis was not in a directly interpretable form. Pas-I provided the

PBG as straight text that had to be reformated into a PBG.

- They did not directly test the model. While the design included comparing the subject's PBG to a trace of a process model, this appears not to have been implemented (Waterman, 1973). In the end, the test was whether the rules based on the model could parse the data with the assistance of a analyst.
- They suffered from poor displays. The intermediate representations in the processing were not directly visible, but were displayed only on command.
- They were naive about the difficulty of parsing natural language. They attempted to do a complete parse without a theory of parsing, using only rewrite rules, and without tying it to a model of the task.

Trace&Transcription. The Trace&Transcription system (John, 1990) was the first and (until Soar/MT) only system explicitly created to analyze protocols with respect to a running process model. It took as input a trace of the transcribed protocol, which could include multiple behavior streams, and a trace from a running cognitive model. In its one and only application these were verbal utterances, mouse button actions, and mouse movements of a user using an on-line help system, and a trace of the Browser-Soar model. These two information streams were semi-automatically aligned; the user would click on the two segments (data and trace) to be aligned, and Trace&Transcribe would add additional cells in one or the other columns to bring them into correspondence.

Trace&Transcription included two innovative ideas that simplify the analysis task and provide more power to the analyst. (a) Treating as a database the data, the model trace, and their comparison. The underlying database system of Trace&Transcription (Oracle) supported queries of where the model matched and mismatched, making available groups such as all unmatched mouse clicks, or all verbal utterances with the word "draw". The database approach also allowed the analyst to group columns of data together, so that they stay aligned as blank cells are inserted to align the model columns with the data columns. (b) Tabular presentation of the data. Each field (e.g., the model trace, the verbal protocol data, the mouse movement data) was represented as a separate column, as were the comparison, comment and time stamp fields. This allowed more data segments to be displayed per given screen size. It also created a visual representation of the alignment of the model trace to the protocol. Rows that are filled all the way across represented correspondences. Blank spaces in the data column while the model column was full represented the model doing more than the subject, and vice versa.

The Card1 algorithm. Card, Moran, and Newell (1983, Appendix to Ch. 5) present an algorithm for finding the longest common subsequence (Hirschberg, 1975) that can be used for aligning two behavior sequences. In their case the two sequences are subject actions from a non-verbal protocol, and actions predicted by a simulation model. In the general case, the sequences can be (a) subject actions from a verbal or non-verbal protocol, and (b) actions predicted by a simulation model, and there can be more than one subject sequence (however we assume in this work that these will always be aligned pair-wise). This algorithm finds the longest common subsequence, that is to say, it finds the largest sequence of tokens (representing matches between tokens in the two streams), such that the tokens in the result have the same order in each sequence.

Other alignment algorithms. Hirschberg presents some additional algorithms (some recursive) for computing the maximum common subsequence that take up less space. The gravest flaw in using these more elegant algorithms is that they do not provide control over which subsequence is returned out of the (possibly) many maximally common subsequences. They may indeed prefer the same one that Card does. They are also more complicated, and speed and space is not of the essence in this application, clarity, readability, and modifiability is. Card1, which uses N^2 space, generally performs the match quickly enough.

Card's algorithm is also based on work by Sakoe and Chiba (1978), which represents the task of aligning a model of speech recognition with the speech signals presented for recognition. This is a

similar task, and can be represented as search with productions (Newell, 1980a). Pas-I (Waterman & Newell, 1971) and Pas-II (Waterman, 1973) proposed (but did not implement) an incremental approach to the match that included backtracking and partial matching. They proposed (Waterman & Newell, 1971) to move both traces ahead by a line, attempt a match, and incrementally advance the counters on the model trace until a match was found. Backtracking was reserved for continued mismatches. Ohlsson (1990) appears to have successfully used this algorithm by hand. A neural net has learned to match a Markov model's predictions to computer interactions in real time (Finlay & Harrison, 1990).

2.3.3 Tools for building and understanding models

2.3.3.1 Process model induction tools

Most process models induced from protocols are created by hand. There has been some work to do this automatically or semi-automatically with machine learning techniques. Semi-automatic generation is done in the event structure modeling domain (a sociological level of social events) by a program called Ethno (Heise, 1991; Heise & Lewis, 1991). It iterates though a database of known events finding those without known precursors. It presents these to the analyst, querying for their precursors. As it runs it asks the analyst to create simple qualitative, non-variablized token matching rules representing the events causal relationships based on social and scientific processes. The result at the end of an analysis is a rule set of 10 to 20 rules that shape sociological behavior in that area. In a sense, the analyst is doing impasse driven programming (i.e., what is the next precursor for an uncovered event not provided by an already existing rule?). After this step, or in place of it, the analyst can compare the model's predictions with a series of actions on a sociological level (a protocol in the formal sense of the word). The tool will note which actions could follow, and query the analyst based on this. Where mismatches occur, Ethno can present several possible fixes for configuration. By incorporating the model with the analyses tool in an integrated environment it provides a powerful tool. It would be a short extension to see the social events as cognitive events in a protocol.

Stronger methods for building models from a protocol are also available. Cirrus (VanLehn & Garlick, 1987) and ACM (Langley & Ohlsson, 1989) will induce decision trees for transitions between states that could be turned into production rules given a description of the problem space, including its elements, and the coded actions in the protocol. Cirrus and ACM uses the ID3 learning algorithm a variant of it (Quinlan, 1983).

Why is automatic creation of process model not done more? These tools look like a useful way to refine process models. These systems do not actually create process models. They take a generalized version of an operator that must be specified as part of a process model. It could be that finding the conditions of operators isn't the hard problem, but that creating the initial process model and operators is. It could also be that it is harder to write process models that can be used by these machine learning algorithms, but these methods should be explored further.

Their lack of use could be simply related to being new software systems. As new systems, they are probably difficult for people other than their developers to use, and they will have to go through several iterations of improvement (like most pieces of software) before they are ready for outsiders to use them. Future work should consider including a machine learning component, for they can help summarize knowledge level information.

2.3.3.2 Tools for understanding and building symbolic cognitive models

While there have been numerous attempts to create general cognitive modeling languages (e.g., Ops, Ops5, Ops83, IPS, Prism, reviewed in Neches, Langley & Klahr, 1987), there has not been many attempts to develop tools for manipulating and understanding the models created for these architectures. When models were small and simple, an additional level of interface was not needed. This is not true anymore. Cognitive models now often contain hundreds of rules (e.g., Browser-soar, Peck & John, 1992) to tens of thousands of rules (Doorenbos, Tambe, & Newell, 1992).

```
P: top-space
O: browse
==>G: (operator no-change)
   P: browsing
   O: find-appropriate-help
   ==>G: (operator no-change)
      P: find-appropriate-help
      O: change-search-criterion
      O: define-evaluation-criterion
      ==>G: (operator no-change)
         P: define-evaluation-criterion
         O: evaluate-evaluation-criterion
         O: generate-evaluation-criterion
      O: define-search-criterion
      ==>G: (operator no-change)
         P: define-search-criterion
         O: evaluate-search-criterion
         O: generate-search-criterion
      O: evaluate-help-text
      ==>G: (operator no-change)
         P: evaluate-help-text
         O: change-current-window
         ==>G: (operator no-change)
            P: mac-methods-for-change-current-window
            O: click-prev-index
            ==>G: (operator no-change)
               P: mac-method-of-click-prev-index
               O: click-button
               O: move-mouse
            O: drag
            ==>G: (operator no-change)
               P: mac-method-of-drag
               O: move-mouse
               O: press-button
               O: release-button
            O: page
            ==>G: (operator no-change)
               P: mac-method-of-page
               O: click-button
               O: move-mouse
            O: scroll
            ==>G: (operator no-change)
               P: mac-method-of-scroll
               O: move-mouse
               O: note-saw-criterion
               O: press-button
               O: release-button
         O: evaluate-current-window
         ==>G: (operator no-change)
            P: evaluate-prose-in-window
            O: compare-to-criteria
            O: comprehend
            O: read-input
            P: evaluate-items-in-window
            O: attempt-match
            O: read-input
         O: focus-on-help-text
      O: modify-search-criterion
      O: search-for-help
      ==>G: (operator no-change)
         P: search-for-help
         O: access-item
         ==>G: (operator no-change)
            P: mac-methods-for-access-item
            O: click-on-item
            ==>G: (operator no-change)
               P: mac-method-of-click-on-item
               O: click-button
               O: move-mouse
            O: double-click-on-item
            ==>G: (operator no-change)
               P: mac-method-of-double-click-on-item
               O: double-click-button
               O: move-mouse
         O: find-criterion
         ==>G: (operator no-change)
            P: find-criterion
            O: change-current-window [...]
            O: evaluate-current-window [...]
            O: focus-on-current-window
```

Figure 2-5: Example output of TAQL space graph.

There have been several disjoint attempts to provide a better interface for Soar on the level of production manipulation and understanding of the goal stack. The two previous graphical interfaces (Milnes, 1988; Unruh, 1986) provided an augmented description of the goal stack that let users click on objects to examine them. These systems did not retain any explicit model of the problem spaces and operators. There have also been three text editors extensions (by Ward, Shivers, and Milnes, respectively) designed for manipulating Soar on the production level. Each included simple commands for starting up a Soar process, editing productions, and loading them. These interfaces were not developed for very long, and were not widely distributed.

There has been only one tool for Soar that attempted to describe Soar's emergent behaviors on the problem space level. Version 3.1.4 of the TAQL macro language for Soar (Yost, 1992; Yost & Altmann, 1991) provides a textual description of the problem spaces and operators that it could recognize from the TAQL constructs making up the Soar model. It did not guarantee that they would be selected, or that the set it found was complete. An example of its output is shown in Figure 2-5.

Other typical cognitive modeling tools based on production systems, such as ACT★ (Anderson, 1983), and Ops5 (Forgy, 1981) come with only a command line interface for loading files and running the system. They do include debugging commands, such as which rules will fire next, but this must be explicitly requested by the user, and a match set is generally not available in a separate display. These systems are viewed as just an inference engine for a cognitive architecture. The production rules that implement the models are created and edited using a general purpose editor, which usually lacks the ability to directly add a new production to the interpreter, and to edit the production in a structured way. All of these production systems lack the ability to describe and manipulate the models on higher levels of organization, such as the problem space or knowledge level.

2.3.3.3 Knowledge acquisition tools

Many types of expert systems are designed to be process models of expert behavior in areas where algorithmic solutions are not available. Expert system development shells (KEE) are designed to build these process models of expertise. Their cousins, knowledge based knowledge acquisition (KBKA) tools, attempt to use an expert system recursively to monitor the process of model building and suggest places that need attention (KADS: Brueker & Wielinga, 1989; KEW: Shadbolt & Wielinga, 1990; Keats: Motta, Eisenstadt, Pitman, & West, 1988; Kriton: Diederich, Ruhmann, & May, 1987; Shelley: Anjewierden, Wielemaker, & Toussaint, 1990; an overview of the field is provided by the Fall 1989 SIGART Bulletin). KBKA tools include many of the features that a tool for testing models would need because their task, to build a process model that produces expert behavior starting with an analysis of the task and verbal protocols, is very similar to testing process models with protocols. Their strength is that they include in their environment, often in a highly integrated way, a process model. In these tools, the process model is the expert system that is being developed. They provide tools to manipulate the expert system, modify it, and run it on the task. They often include graphic depictions of their declarative knowledge (e.g., Keats: Motta et al., 1988), but they rarely, if at all, provide visual descriptions of the processes and the process knowledge.

Knowledge acquisition environments sometimes include the ability to tie verbal protocols or other texts to various facets of the model, showing where a feature came from, and serving as a note that a segment has had its knowledge extracted. Their task is only to create a model that performs the task, not to validate the model's performance against the input, so they all completely lack the ability to measure the comparison. As a group, when coding protocols they also make a fundamental misunderstanding about the comparison of similar levels in the protocol and model. They always code the protocol in terms of the static structure of the model, either as rules or data structures. When an expert is talking about rules to apply, that is appropriate. When the same tool is used to code verbal protocols given by an expert while they are performing the task, this coding is inappropriate. In terms of Figure 2-2, they are comparing the knowledge of the model to the process data of the subject. Like many cross-level comparisons, it is incorrect and only approximate, but in practice it serves them well. One system, Kriton (Diederich et al., 1987), claims to be able to do this automatically. How well it

does this is not clear from the short technical report.

2.3.4 Summary of useful tool features

Reviewing these tools for performing various subsets of testing process models with protocol data suggests several guidelines for future integrated tools.

1. Current automatic testing approaches carry too many constraints for general use. While it is possible to automatically compare a process model with data, several intelligent tutoring systems do so with non-verbal data, it does not appear to be currently possible with verbal data because of the difficulty of doing general natural language parsing. Natural language parsing is necessary to match verbal protocols to the process trace. Only two systems attempt to compare verbal protocols to models automatically. It is not clear that the parser in Pas-II is powerful enough to be applied in a truly automatic sense. The details provided on how the Kriton system does its parsing are not adequate to judge its performance.

   Systems that otherwise do automatic analyses must limit their general applicability by taking on one or more constraints to avoid natural language, such as using only well tested models and simple tasks that use a limited subset of language, or working with discrete data. Future general tools can only provide semi-automatic analyses until parsing technologies are further developed.

   The task of testing process theories with verbal protocols may present a unique opportunity to push natural language parsing forward. Unlike general parsing situations, the model being tested provides a strong theory of what will be said. The directly relevant knowledge structures needed for parsing, presumably the data structures used to perform the task, are available and updated by the process model as the task unfolds. Using multiple behavior streams to fit to the model would further restrain the parsing task.

2. Automatic and semi-automatic features aid reliability and speed. Tools that provide assistance doing the analysis that are automatic (completely autonomous) or semi-automatic (small user initiated analyses, or semi-correct analyses checked by the user) are highly praised by users, even if they are not always tied to the theoretical constructs being tested. People, even dedicated ones, do not like doing this type of work. Analyses done without tools that provide assistance are not repeated as often. The most automatic systems, the General Inquirer (Stone, et al., 1966) and the Anderson tutors, truly make the analyses they do routine.

   Semi-automatic tools may actually have been used more often because fully-automatic analysis requires the theoretical model of the analysis to be complete. The system can do this by having a weak but quite general model, such as the General Inquirer (Stone, et al., 1966), or by having a well developed model such as the Anderson tutors have. If the system does not provide all the specifications for the analysis, then the user must create them. If too much specification is required from the user, the system might not be used, which may be what happened to Pas-II.

3. An extendable word processor and a database facility are required. Most tools incorporate word processor functions to do such things as annote segments, correct transcription errors, and enter and edit labels. Systems that do not incorporate a word processor (such as PAW, Fisher, 1991), assume that one is available in the environment.

   While the broad approach to protocol analysis and model testing can be specified, in the end these are fluid tasks. A set of analyses must be provided, but additional, similar

analyses will be necessary. Systems should provide a macro language and interface to help automate repeated actions, to create modifications and extensions, and to integrate with other tools.

Simple database facilities are also required. These facilities need not be extensive, but the basics must be provided. Only Trace&Transcription (John, 1990) and some general tools appropriated to the task incorporate a complete database system (Oracle). Most tools provide some support in this area. Nearly all the systems support adding additional data fields to segment records. Those who do not suffer for it. For example, those that do can be expanded to include their verbal protocol as a separate field, even if their initial configuration does not. Simple aggregations by types is another very desirable database feature. These simple analyses appears useful for checking the ongoing analysis for correctness. More sophisticated systems include more extensive analyses built in, such as sequential lag analysis. The more advanced database functionality of listing segments by queries appears useful, although few systems currently support this.

4. Tabular displays present more data. Figure 2-6a shows a record based display, implemented as part of an initial version of Soar/MT. It is representative of the record based displays of most tools. Figure 2-6b shows a tabular based representation of the same data. The tabular approach represents a segment as a row, with its fields placed in separate columns, like a spreadsheet. This approach better supports the appropriate visual operators (Larkin & Simon, 1987) for finding the context of a segment and its associated fields. By only using one line per segment, and putting the labels only once at the top, more data can be displayed than in a multi-line record approach. This tabular format also supports the approach analysts have used when working on paper (Newell & Simon, 1972; Ohlsson, 1980; Young, 1973). Trace&Transcription and some of the general purpose tools support a tabular based visual layout. For a telling (and atypical) supporting example, see Newell and Simon (1972), Appendix 6.1 and its notes two pages later. The distribution of information across several pages makes this protocol hard to read.

5. Integrating the model support analysis. Tools should keep the models being tested close at hand. If a tool does, its manipulations can be directly based on the model being tested or built. This includes automatic analyses based on the model (such as features supported by data, features unconnected to other features). Having knowledge of the model is required for helping the analyst through automating the tasks or by providing smart features that partially do the task. This representation must be explicit. Several systems that use models (e.g., Pas-I and Pas-II) include them only implicitly in production rules. This precludes the tool from using the model codes the rules represent without running the model to find them.

   Systems that can automatically offer codes to assign to data points represent the weakest form of this ability. They know the names of the model components, but that is all. Most systems, including general purpose ones, either directly provide this low level of model manipulation or can be modified to do so.

   Pas-II represents an unfortunate position with respect to incorporating a model. It did not directly include a model that could be used to do an analysis. The model being analyzed was only available implicitly in the productions that users added to do the analysis, and not directly available. Pas-II could manipulate the data with respect to the model, but only if the user supplied these manipulations as additional productions based on a model external to Pas-II. It did not check directly to see that all the operations in a model had been supported by data, or suggest operators to code with.

   The model induction tools (ACM, Cirrus), and most knowledge acquisition tools can

directly access their model as a knowledge source for analysis. Indeed, in expert system development knowledge-based knowledge acquisition (KBKA) can base the analysis directly in terms of the model. Tools that include declarative model structures (e.g., Kriton and Keats) can reference the model directly. Having the model directly at hand provides KBKA with strong support for coding and data aggregation, but these tools do not completely incorporate comparisons because they have been designed for building models, not for testing them.

In many ways Trace&Transcribe is the best extant tool for testing process theories. Its major drawback is that it cannot reason about the model it is testing. Trace&Transcribe also does not explicitly represent the Soar models being tested, and the models are also not available from the current Soar implementation (5.2). After the alignment of the subject protocol with the trace, there are no further analyses that it can carry out on its own based on the model, such as noting which operators were supported, or where operators were not supported.

6. An interface is required for manipulating and understanding the model. The lack of development of interfaces and display representations for manipulating and understanding symbolic models can be contrasting with the development of tools for PDP models (McClelland, Rumelhart and the PDP research group, 1986; Rumelhart, McClelland and the PDP research group, 1986). PDP models can be difficult to interpret directly; their structure is implemented as an array filled with real numbers representing the connections between nodes. PDP software has mitigated this problem by letting modelers manipulate systems on the level that they think of them. Nearly all systems now allow the user to create models by drawing their nodes and connections (O'Reilly, 1991). The diagrams created this way are also used to describe the model's performance over time, such as the changes in the connections from learning are often visually displayed during a run (McClelland & Rumelhart, 1988). Displays to illustrate the importance and meaning of the connections have also been developed (Hinton & Sejnowski, 1986; Kolen & Pollack, 1988; McClelland et al., 1986; Rumelhart, et al., 1986; Touretzky, 1986).

2.4 Measures of model to data comparison

This section reviews the measures that have been used by analysts to improve and describe to others the fit of process models to data. Measures for comparing sequential data will primarily be examined, but measures for comparing aggregate data will be included when appropriate. A shorter, and slightly different view of measures useful for model building is available in Sanderson et al. (1990).

The review starts off by deriving what is needed from previously published and newly presented criteria for evaluating these measures. Taken together the criteria indicate that four different types of measures are necessary. These four types of measures include (a) a global measure of where the model mismatches the data, (b) a simple measure of fit to provide local guidance for improving the model, (c) a measure indicating how well the model will perform in the future, and (d) a measure indicating the degrees of freedom in the model. Some of these measures must also be persuasive to others, so this is listed as the fifth criteria. The ability to describe the behavior of the model in general terms also turns out to be important in comparing models to data, even though it is not itself a measure of fit. The most influential need, for measures to indicate where the model does not fit the data, particularly makes strong recommendations about which measures are useful.

The measures that have been used to evaluate the fit of process models can be categorized into four types: (a) non-numeric descriptive measures of the general fit, (b) numeric descriptive measures of the general fit, (c) measures of rule (or component) utility, and (d) measures based on inferential statistics. Previous uses of each of these types will be described, and they will be evaluated with respect to how

2-6(a) Example record-based display of model trace to data comparison.

```
1 ----------------
TIME: 12400   VIS:       TYPE: too-short
VNUM: 1   DURATION: 725  VERBAL>: I believe
MNUM:     MBANUM:        MOUSE>:
DC:      B EVID FOR:
  V EVID FOR:
  M EVID FOR:
  M REQUIRED:
Comments:
2 ----------------
TIME: 12406   VIS:       TYPE: v-coded
VNUM: 2   DURATION: 351  VERBAL>: write
MNUM:     MBANUM:        MOUSE>:
DC: 15   B EVID FOR: O: o82 (generate-search-criterion)
  V EVID FOR: O: o82 (generate-search-criterion)
  M EVID FOR:
  M REQUIRED:
Comments:
3 ----------------
TIME: 12409   VIS:       TYPE: v-coded
VNUM: 3   DURATION: 2210 VERBAL>: write
MNUM:     MBANUM:        MOUSE>:
DC: 21   B EVID FOR: O: o82 (generate-search-criterion)
  V EVID FOR: O: o82 (generate-search-criterion)
  M EVID FOR:
  M REQUIRED:
```

2-6(b) Example tabular display of model trace to data comparison.

```
T  Mouse actions  Window actions  Verbal  ST # Mtype MDC DC                    Soar trace
-+--------------+---------------+--------+-+-+-----+---+--+-----------------------------------------+
 0                               I believe  v 1 short
                                                          0 G: g1
                                                          1 P: p4 (top-space)
                                                          2 S: s5
                                                          3 O: browse ()
                                                          4 =>G: g19 (operator no-change)
                                                          5   P: p26 (browsing)
                                                          6   S: s39 ((unknown) (unknown))
                                                          7   O: find-appropriate-help
                                                          8   =>G: g43 (operator no-change)
                                                          9     P: p50 (find-appropriate-help
                                                         10     S: s59 ((unknown) (unknown))
                                                         11     O: define-search-criterion
                                                         12     =>G: g65 (operator no-change)
                                                         13       P: p72 (define-search-criterion)
                                                         14       S: s79 ((unknown))
 6                               write      v 2     v 15 15       O: generate-search-criterion ((write))
 9                               write      v 3     v 15
13                               write      v 4     v 15
  M(+x) (R of prog win)                              B4
               mouse line to pointer
                                                         16       O: evaluate-search-criterion
                                                         17     O: define-evaluation-criterion
                                                         18     =>G: g103 (operator no-change)
--**-emacs[SHAMO.SOAR]: example-types.spa    A36 ManUp  <H>  (SPA) ----Top-----------------------------
```

Figure 2-6: Example displays for comparing the model's predictions with the data.

they serve the identified measurement needs. This section ends with a summary of the most useful measures.

2.4.1 Using criteria to develop a set of measurements

What are we trying to find out with these measures of models? Listing a set of criteria for each measure is a way to answer that question. Table 2-4 displays the eight most important criteria that have been put forward for desirable model measures. Taken together the criteria determine which measures to use. So, what are the questions that are typically asked about the data and the model? These can be grouped into five basic requirements discussed in order of importance. There appears to be two general questions that people ask: How well does a given model fit a given set of data?, and How can we tell if this is significant match? These questions will basically be rejected in the following sections and replaced with two others.

Table 2-4: The five major types of measures of model fit and the criteria supporting them.

(Criteria in []'s are later rejected.)

1. Globally showing where the model mismatches.

- The analytic testing criterion. The measure should indicate where the model does not fit the data (Grant, 1962).

2. Locally showing where the model mismatches.

- The inaccuracy criterion I. The measure should decrease for every false prediction (Grant, 1962).
- The accuracy criterion I. The measure should increase every time the model fits the data (Priest & Young, 1988).
- The inaccuracy criterion II. The measure should decrease each time the model does not fit the data (Priest & Young, 1988).
- [The accuracy criterion II. The measure should increase for every correct rejection (Ritter, this thesis).]

3. Knowing how the model will perform in the future.

- The prediction criterion. The value of the measure obtained from the data sample should provide an unbiased estimate of the value to be obtained from a larger sample (Priest & Young, 1988; Grant, 1962).

4. Representing the Fit vs. complexity tradeoff.

- The parsimony criterion. The measure should decrease for each additional variant procedure added to the system to account for the data (Priest & Young, 1988).

5. Being persuasive.

- [The numeric value criterion. The measure should return a single number falling in a predetermined range (Priest & Young, 1988).]

Globally showing in terms of the model where the predictions mismatch. In his seminal paper, Grant (1962) notes that scientists are not really in the business of testing theories just to put a stamp of approval on them, which is often presented as the questions of how well does the model match, and is it significant? Scientists are more like a parachute maker who wants to make a better parachute. The parachute maker tests parachutes to find out their weaknesses, and where to make them better. As scientists conducting ongoing research "[the theoretical scientist] is not accepting or rejecting a finished theory; he is in the long-term business of constructing better versions of the theory." Grant calls this approach analytic testing, it is "designed to tell me as much as possible about the locus and cause of any failure in order that I may improve my product." Analytic testing is then computed to tell where to improve the model, and not merely to provide a stamp of approval.

Usually the mismatches pointed out will be small errors, such as the order of performance of subtasks. Sometimes they will be big errors, such as completely different approaches to tasks or whole competencies not provided in the model. In either case we need to know the location and extent of both sizes of mismatches, and a way to see any systematic patterns in the errors so that we can fix the

model. Comparing the model with the data is thus two tasks, noting where the model is consonant with the data, but more importantly, noting where it is not consonant and needs to be improved.

In striving to predict more of the data, process models become more falsifiable, generally considered a good thing for a model. Popper (1959) argues that it is better to be falsifiable so that it can be more easily discarded if it is wrong, as it will be. What Popper should have meant, and what being falsifiable means for us as theory builders and improvers, is that theories should be falsifiable so that you can see where to improve them! Being falsifiable for us means not only that the predictions be concrete, but that the model make many of them, and that they be as detailed as possible, even including sequential information, so that where the model is wrong we can more easily and often tell where we need to improve it (Newell, 1990, p. 14).

Least we forget though, we also need to be able to characterize where the model performs well, so that we can know where to use it. In the parachute maker's terms, we need to know what objects and at what heights are the best places to use our parachute. Knowing the tasks where the model performs well is also useful for focusing attention on and characterizing tasks where it performs poorly. Finally, knowing where a model performs well will also be necessary for comparing the model with other models.

Locally showing where the model mismatches. In addition to global measures of how well the model fits the data, a local description of where the model mismatches is necessary to implement local improvements. This need not be a separate measure, but could be incorporated into a global measure if the global measure was sufficiently convenient.

The four accuracy and inaccuracy criteria essentially represent the four measures in signal detection, those of hits (subject matches model's prediction), misses (subject action not predicted), false alarms (model's prediction not matched), and correct rejections (model and subject correspond on inaction). These measures are not global measures of where to improve the model for they are only a count of places that could be improved. Strict numerosity is not required, merely that the measure gets "better" somehow for matches, and not only gets worse on mismatches, but points out where they occur.

Fit versus model complexity trade-off criteria. The final two criteria for measures are related to parsimony. The first is that the measures take into account parsimony, how many degrees of freedom were used to account for the data? Strictly speaking, this is not necessary for finding out where to improve a model, but for knowing when to stop improving its fit to a given data set. Degrees of freedom are normally reported separately (e.g., Chi-squared tests, T-tests, F-tests), and we will encourage that here as well. How many variant procedures have been added to the system to account for the data should be indicated, but other measures need not incorporate this directly.

Knowing how the model will perform in the future. Given this view of scientists as model builders and users rather than as model certifiers, the other important criterion for measuring a model is "How well does the model make predictions for the future?" This will be the main measure used for public display.

Being persuasive. One of the problems that has plagued researchers in this area is that they want to be able to persuade other scientists through their measures that their model fits the data well. This is a real need, and has distorted the choice of measures. In the past, various statistics from experimental psychology have been incorrectly applied in an attempt to do this by proving the models different from chance or not different from the data. These were never convincing, and rightfully so for they are, respectively, weak and incorrectly applied tests. The real need is to show how well the model will fit other data, not how likely it was to fit the current data as well as it did.

The numeric value criterion put forward by Priest and Young (1988), that the measure should return a single number falling within a predetermined range was to ensure that different micro-theories could be ordered for a particular technique. And it expresses the desire to provide a stamp of approval that Grant (1962) notes exists. This has to be rejected as an absolute requirement, for it assumes that

process model fits should be an ordinal number. And they cannot. Ordinal numbers cannot indicate multiple places where a model needs to be improved. Rejecting this also frees us from trying to find a way to combine all the accuracy and inaccuracy criteria into one measure.

If one was interested in incrementing a single numeric measure whenever the model fits the data, strictly speaking, the measure must also be incremented when the subject does nothing and the model does nothing as well. Correct rejections make sense when there are fixed items for responses, but it makes less sense when modeling a continuous process. It would be a difficult problem to decide on a reasonable method for enumerating all the places where the subject and model did nothing and to increment a measure based on this. This criterion can be rejected because the requirement of only needing to know where models mismatch frees the measure from including all the places where it does match.

Summary of required measures. The ability to show where the model globally and locally mismatches, and the ability to predict the quality of future performance based on where the model performs well are the three most important measures. We would prefer a unified measure, but will have to use several to cover the ground, particularly since different models can mismatch the data many different and subtle ways. Using multiple measures is acceptable because there are many compatible ways to improve something — we are not trying to put a stamp of approval on the model, where a single standardized test would be preferable. These measures do not have to result in a single number, and the numeric criteria of increasing the measure for every hit and so on, is taken loosely to mean to influence the measure appropriately in each case.

Presenting the measures that are used to improve the model and one that is a measure of the degrees of freedom in the model, if presented clearly, will hopefully convince others. These alone do not have to. The analyst also has the predictive power to report (which has been computed as the analysis has gone along), or to take home and use in applications that may also prove the model.

2.4.2 Description of measurement inputs

This subsection describes the possible inputs to the measures, including the two information streams, and the types of results of comparisons on the data level.

The two traces. As diagrammed in Figure 2-2, when a process model is run, it generates a trace of its external task actions and internal states and operators. These can be labelled M_1 through M_y. Each M_j is a trace of model actions on the appropriate level. It can be the rules that fire, or working memory elements, operator applications, or knowledge that has been applied. The choice depends on the theory level committed to and on the level of subject behavior available. If the model is based on rules rather than operators, then the trace could also include or consist solely of rule firings. In Soar models in general, and particularly the ones examined here, it is operator applications. Each model action has a simulation time associated with it. The model stream will include a fixed number of token types, specified ahead of time as part of the model.

On the other information stream, the subject generates a series of actions, S_1 through S_x. Each action should have a time stamp (in seconds or ms) associated with it. Not having a time stamp will preclude some of the analyses. As they are sequentially ordered data, they are a protocol. Different modalities, such as verbal utterances and eye movements, each represents a different information stream, but order across modalities is preserved. These information streams may contain an essentially unlimited number of different types of words.

The measures will be computed after the subject's actions (S_1 to S_x) are interpreted and aligned with respect to the model's predictions, (M_1 to M_y). As a simplifying assumption, each correspondence will be a full one; partial matches on the segment level will not be allowed. Measures may take into account multiple episodes for a subject, or multiple subjects, but the emphasis will be on fitting a single episode. Each measure or representation must represent each of the types of matches shown in

Table 2-5.

Table 2-5: Types of correspondences between the model's predictions and the data

1. Uncodable subject action. There may exist subject utterances too short to code or outside of the model being tested ("Hmm", "nice day"). If they are clearly outside the model, the analyst may wish to discard them from later analysis. Sometimes it will be useful to carry them along as comments because they serve as way posts, or they may be found to be data with respect to a more complete model. Measures and displays should be able to handle them as null points.
2. Uncodable model action. There may exist objects in the model's trace that represent internal state and operations. Not all model actions can or will be matched in a verbal protocol. Measures and displays should be able be able to handle these too as null points.
3. Simple hit. A subject's action and a model's action correspond one-to-one.
4. Multiple subject action hit. A single action in the trace may match multiple subject actions. This could be caused by a high level operator in the trace that represents actions larger than a typical segment, or else takes multiple descriptions or matches data in multiple modalities ("I'm pressing the space bar" and the keystroke action <space-bar>).
5. Multiple model action hit. A single subject action may correspond to multiple actions in the model. This is possible if the model performs the task on a finer level of detail than the data provides, or if the segment is incorrectly segmented. The analyst should consider splitting the segment when this occurs.
6. Miss. The subject action is not matched by a corresponding model action.
7. False alarm. The model produces an action that the subject does not perform. Overt task actions represent the most egregious example of this, and must be penalized. Unsupported internal actions of the model should not be penalized, but must be supported some other way. This can be done through appeals to necessity, or aggregate data. It may also be the case that portions of the model cannot be directly supported, but are required by the architecture.
8. Crossed in time. The model and subject actions would match, but are performed out of order with respect to other actions within the same modality, or across modalities. Actions matched out of order between the two streams will pose a dilemma of how to score them. A strict position is possible, that of only allowing monotonically increasing matches within a single information stream. This will be assumed within each behavior stream of a subject. Task actions, for example, clearly have an important ordering to themselves. Ericsson and Simon's (1984) theory of verbal protocol production asserts that verbal utterances are produced in the order that they enter working memory, and we will assume this for other protocols that report on internal state as well. This sequentiality matching requirement cannot be applied between multiple information streams until it can be supported theoretically or empirically.

When the model does not match the data. Process models will fail, they will make predictions that are not supported, and the subject will do things that the model did not predict. There will be subjects that the model cannot be said to model at all. That this will happen has been noted since their first use (Newell & Simon, 1972, p. 197). It is not the end of the world. No model gets all the data all of the time. Not getting verbal utterances (categorical data) matched may seem worse, but that is only because it does not look like the noise in numeric measures that we are used to seeing.

Table 2-6 notes the several approaches that have been used to deal with mismatches. Which approach to use will depend on the desired result of the analysis and the tractability of the model and subject. One approach adds new requirements to the modeling tools, and one suggests a useful constraint for the subject's experimental situation. A combination will often be required. It may be appropriate to minimizing the effect of mismatches in different areas in different ways. For example, the experimenter may choose a novel task domain in order to keep the effect of previous knowledge low, so that the mismatches are caused solely by the model in the area of interest, say problem solving.

Table 2-6: Ways to deal with mismatches

- Avoid mismatches (if you can).
- Reset the model: Conditional Prediction.
- Reset the subject.
- Reset the data.
- Reset the interpretation assumptions.
- Do nothing.

Avoid this (if you can). The cleanest and most desirable way to deal with mismatches is to avoid them (and this is the way your mother would tell you do deal with this, at least mine would).

- Choose regular task domains, where a model is available and it already predicts that the subject will be well behaved or driven by the task. The tasks must also be those where the subject's behavior does not depend on previously learned knowledge invisible to you. The common choice of novel problem solving domains reflects this constraint (e.g., the Tower of Hanoi). The drawback of this approach is that it encourages the models to stay close to home, to not attempt to model the unmodeled.
- Do small tasks at a time. By defining the starting state for the subject more often through the task definition, it is intrinsically easier synchronize subject's actions with the model's predictions, and it provides a greater number of tests of the predictions (although shorter sequences are tested). For example, studies of subtraction problems have benefited from this data set feature.
- Provide a good simulation of the environment. Providing a realistic model of the environment will allow model to come back on track. If the environment provides fixed responses to any model action (e.g., on the 1st mouse action, display the first menu), the model must match the subject to go on with comparison (Kieras, personal communication, May 1992).
- Repair the model. The mismatch often tells you something; so go and fix the model so it mismatches no more.
- Reexamine your interpretation of the model and its predictions if its predictions are not automatically derived. The interpretation of the data with respect to the predictions should also be examined.

Reset the model: Conditional Prediction. The most common approach for dealing with mismatches that start truly divergent behavior is to perform condition prediction (Feldman, 1962). Upon divergence in behavior, condition prediction specifies that the model's prediction that went awry gets noted as a mistake, and then the model gets reset to the state of the subject at that point, and the model makes a prediction from that point. The summary measures must then include the number of mismatches as a reported result. The errors a model makes result from either an incorrect model, or an incorrect

specification of the initial state. Conditional prediction allows these two types of errors to be treated separately (it does assume that the subject's state can be determined so that the model can be set to it). This approach has been used quite often in teaching programs that learn (Samual, 1959) and testing cognitive models (Feldman, 1962; Newell, 1972; Newell & Simon, 1961; Newell & Simon, 1972, Ch. 12; Newell, Shaw, & Simon, 1959; Samuel, 1959; Young, 1973).

This approach is theoretically justified. The actions mismatching after the first mismatch are consistent with it, given a different starting state (the state that results from the subject and model applying different operators), the subject and model will often end up in a different state, and choose different paths after that. There will be areas where this assumption is not desirable, where the length of the sequence of correct predictions is important, but this work does not take up that assumption. This makes the most sense when the model provides a set of equally likely actions or a set of actions by likelihood (Anderson, Conrad, & Corbett, 1989).

As noted by Feldman (1962), conditional prediction has many advantages: (a) It ensures that the errors at any point are caused by new mistakes rather than carrying on from old ones. (b) By reconsidering after each mismatch, conditional predictions provides a framework for testing each component of the model. By resetting the model at mismatched intermediate states, later parts of the model can be tested even if the initial performance is not perfected yet. (c) By removing the number of errors caused by previous errors, the model's predictions can be tested more often.

In a production system model, the model can simply be reset (e.g., Feldman, 1962; Newell & Simon, 1972), or it can be modified by inserting simple productions that correspond to each divergence (e.g., "if cycle = 3 then set operator to be add-3"). If the mismatches are truly fixed points of departure, and are represented as such, they will never be used again, and it is not worth attempting to generalize them. If several mismatches appear similar, it may be worth attempting to generalize them, providing a real fix if the problem occurs across subjects. (e.g., "if environment-attribute = value1 then set operator to be add-3"). This is the process that ACM (Langley & Ohlsson, 1989) and Cirrus (Kowalski & VanLehn, 1988; VanLehn & Garlick, 1987) attempt to automate. If the mismatches are the result of individual differences between subjects, then the system becomes a basic-model plus an individual difference modification (Miwa & Simon, 1992). One cannot necessarily tell when the attempted fixes are inserted which they will turn out to be. That they are correctly general can only be found out with more data, particularly, data representing the same situation.

Reset the subject. Given a suitable domain and social environment, the subject can be reset instead of the model. This ensures that the subject does not stray far from model. This is not generally applicable because it requires special domains where a task environment has license to reset the subject's state (e.g., tutoring). Therefore it is mostly used for closed analysis with well developed models. Anderson and his group, for example, do this in their tutors. The subjects are actually taught by resetting them.

Reset the data. Sometimes the mismatch will be so gross, and the model so persuasive, that data itself must be questioned. In areas outside psychology, doing this can be quite common (Heise, 1987; Heise, 1989). There are a few examples showing that this is possible (e.g., Feldman, 1962). And in the analyses reported in Chapter 7 we found two transcription errors and one interpretation error based on the model's predictions. They were noted as wrong, and reset. While it is possible, it must be applied selectively and carefully. Over-belief in the theory is a mark of zealots, and a theory can end up untested.

Reset the interpretation assumptions. In testing the predictions of any theory, assumptions are made about the mapping between the theory's predictions and the real world. In this relatively young area, how to map the predictions of cognitive actions to real world events such as verbal protocols and hand movements, is not yet well developed.

Do nothing. There will be areas where the model will not match, and the final approach is not to do anything about the mismatch except note it, and continue the analysis from the $n+1^{th}$ segment

(Ohlsson, 1990). This is, of course, the least desirable approach. There are two types of unmatched data that could be ignored. The most natural to ignore are actions that are outside of the scope of the model's coverage. Nose scratching, laughter, and meta-comments should not count against the model, but the amount of this noise should be included in summaries. The other type of unmatched data are subject behaviors that should be predicted by the model. These are more dangerous to ignore. The sequential and dependent nature of the actions will lead this approach to very poor performance, particularly if it is used exclusively. This is done for whole subjects when their data are not used because it is "too chaotic to analyze."

A note on resetting the model. Each of the measures that will be discussed are based on the raw comparison between the model's actions and the subject's actions. The model and the subject will mismatch, that is what the measures are there to measure. However, if the two traces completely diverge, the comparison stops being useful. Without local adjustments to reset the model or tutoring instructions to reset the subject to bring them back into correspondence, all of the measures will provide less information on how to improve the model. The model will start the next action with a different initial state than the subject does. After a divergence, the model ends up being tested with less of the data because the model's actions are based on a different set of initial conditions after the time of the divergence than the subject's are based on. All the measures, but for measures of parsimony, will assume that the model can be resynchronized with the data when needed.

Initial measures. In addition to the aligned traces, there are several simple building blocks based on the model's actions or the subject's actions that are used by more elaborate measures. The measures of the subject's data include the number of segments, the length of the protocol in words and seconds, and thus the word rate, as a check on the quality of the protocol. Similar measures are required from the model, including the number of model actions, a measure of the internal time of the model, such as the number of production firings, operator applications, or other constructs. Taken together, these two sets of measures show the temporal density of actions for support. Additionally, analyses of parsimony will require the number of rules or separable components in the model.

2.4.3 Non-numeric descriptive measures

The first set of measures examined are those that provide direct, non-numeric descriptions of the model's performance with respect to the data. They do not provide predictions of the model's future performance, but are used to tell globally where the model mismatches the data and could be improved.

Informal qualitative comparisons. The test can be performed in several ways. The most straightforward measure that has been used is to present the model's predictions and the subject's actions together but not necessarily aligned (Feldman, 1962; Newell & Simon, 1961). The analyst asks himself (or others), Do the two information streams appear to be different? When it was first proposed, it was put forward as a type of Turing (1956) test.[1]

After the subject trace and model trace have been aligned, it is possible to simply display the correspondences in a table or diagram. This comparison also has some bearing on the utility of each operator or rule that generates the actions, but this is limited by the viewer's memory.

The simplest way to present the correspondences is to just present the two information streams aligned side-by-side. It is often used by analysts in their work (e.g., Appendix I of Chapter 7 in this document, Appendix 6.1 in Newell and Simon (1972), and Appendix B of Young, 1973). Portions of the alignment are sometimes used when reporting results (Young & O'Shea, 1981). The model's trace is often compared to protocols in a simple, informal way (e.g., "This trace is remarkably similar to

[1]It actually is the popularized form and name of the Turing test. In the original Imitation Game, Turing calls for the test to be between a man imitating a women and a machine imitating a woman.

protocol D", Luger, 1981, p. 70). If the comparison is done on a high enough level, all the correspondences can be presented (Johnson, et al., 1981). This level of detail is necessary for local improvements. By presenting a sufficient number of rows together, some context and flavor of the match can be obtained.

Process models are not normally implemented to the level of producing verbal output, but in an interesting extension in the direction of the popularized Turing test, Ohlsson (1980) added to his model of linear syllogisms the capability of producing talk aloud trace. The model's output appeared nearly human when compared side by side with a subject's verbal utterances. Seeing them together and being able to note the similarities made the model more believable. In this case, it was interesting to note how mechanistic the subject's speech was.

This measure weakly supports showing where the model can be improved. Where model actions were and were not matched can be found through visual search, but a more global view is often needed of which model actions or knowledge structures were matched and how often they were matched.

Informal comparisons are most often applied by reporting aggregate descriptions of the subjects' and model's qualitative (e.g., Larkin & Simon, 1981, Simon & Simon, 1978) or quantitative (e.g., Carpenter, Just, & Shell, 1990) behaviors. No formal comparison is made, the two behaviors are just described, and the reader is left to judge on their own the significance of the match.

This can be a useful measure, because in addition to being able to test that the model can perform the task on a basic level, it also brings to bear all kinds of unspoken constraints on the task performance. It almost always can tell an observer something about where the model could be improved, particularly the global ways in which the model does not fit the data. In a certain sense, it is used informally by all researchers when examining a model's initial performance. It will indicate global and local places where the model mismatches. It will probably not provide a reliable measure of future applicability. If being fooled as part of a Turing test convinces, then it is also one way to be persuasive.

While this measure does not commit any errors, this type of Turing test fails to be complete enough to serve as the only measure of a model. Knowing more than it can tell about the fit is generally required. It also fails on its own to highlight where the model fails to match; it requires an intelligent observer. The observer also has weaknesses, and will not be able to bring all the constraints in the data to bear. Finally, it fails to make predictions about how well the model will perform in other situations.

Contingency tables. The simplest way to summarize the comparison is to aggregate the correspondences by model action in a table. The model actions that would match subject's actions and the actions that actually occurred in the model can be compared with an N-way contingency tables (e.g., Newell & Simon, 72, Figure 6.8). In this type of table, the row headings are rules or operators that would match the subject's actions, that is, could fire, and the column headings are rules that did fire. The firings and possible firings are totaled up from each segment, and the totals placed as counts in each cell. This is a useful diagnostic aid. Cell counts on the diagonal indicate that model components could and did match the subjects behaviors. Otherwise the cell counts represent misses for the rows and false alarms for the operator represented by the column. The counts can be aggregated over subjects (Larkin, 1981; Larkin, et al., 1980) or over episodes of a single subject (Newell & Simon, 1972).

They are useful diagnostic aids because they start to summarize the model's behavior with respect to the subject's, and start to suggest which parts of the model could be improved. It major drawback has been that it is difficult to compute by hand. It also does not directly indicate where the mismatches occurred. It would be more useful if the comparisons making up the cell counts were accessible as a set.

Visual displays of the correspondences. There are two particularly interesting displays for presenting the match between the model and data. This type of analysis (without a graphical display) is also supported in some knowledge acquisition tools (e.g., Kriton: Diederich et al., 1987; Keats: Motta et al.,

1988). They allow the analyst to manually (or automatically) indicate by showing for each knowledge structure the matching subject segments. No cumulative measures or complete listings are provided however.

Figure 2-7 shows the first display. Peck and John (1992) created a display that graphically depicts the correspondences between the model's structure and predictions, and the subject's actions. It is an operator application support graph that depicts the operator application order of a process model and a visual description of which actions of the model were successfully matched by the data, and which parts are either unused or are not supported by the present data set. This diagram helps the analyst answer several questions: What actions of the model were supported? At what portions of the episodes do these appear? What proportion of the time does the subject give verbal support when they could? If there is an order to where the model does and does not fit, this diagram may be able to suggest where it is occurring, and what model elements might be involved.

This graph has two shortcomings. The largest shortcoming by far is that it currently must be generated by hand for each episode and each time the model changes. This makes it too expensive to use routinely. The other problem is that while it tells us directly where the model is wrong, that is, what protocol segments are not predicted by the model, it could indicate more clearly how the model is wrong.

Figure 2-8 shows the second interesting display. In this display, Sakoe and Chiba (1978) present the match between the predictions of a speech recognition algorithm and the actual speech. The time course of the correspondence is given as a warping function, which can be used to directly compare the time course of the subject and the time course of the model. This graph is used to compare sequential, time dependent predictions, except they are working in a model of simple speech production instead of a cognitive model. The time scales used here are relatively smaller, 100s of milliseconds, than most of cognitive models that operate in the 10 to 100 s domain. This will be used to build a display in a later chapter.

This graph is related to one by Just and Carpenter (1992, p. 140) where they compare the time course of the match between the cognitive model of reading versus the subject's actions. The difference is that they put each time course on different graphs, which makes the comparison more difficult.

Summary. The general performance of a process model must be described in order for it to be understood, so the qualitative descriptions that make up the popularized Turing Test must be performed for every model, while both building and testing the model, and when describing it to others. This test requires a skilled observer, so it will be the last measure to be automated, when we can create machines not only intelligent enough to pass the Turing Test, but to judge others.

Graphic displays of correspondences are necessary and useful, as well as the summary tables showing how much of the model is used and matched by the data. They constitute theoretical predictions of the importance of the various model parts.

2.4.4 Simple numeric measures

Simple descriptive measures of the model's correspondence to the data can be computed. They are often the first step in an analysis. They are useful for measuring local improvements while developing the model, and as simple summaries of more detailed comparisons. The number of subject actions matched to model actions can be stated in signal detection terms as noted earlier. Hits are places where the model and the subject perform the same action; misses are situations where the model does not perform an action that the subject performs, and false alarms are actions that the model performs that the subject does not. These simple measures are often reported in model descriptions, They have included the number of actions to do the task for both subject and model (Simon & Reed, 1976), the percentage of subject actions matched (e.g., Larkin, et al., 1980; Larkin, 1981, Young & O'Shea, 1981), and the percentage of model actions matched (Larkin, 1981; Larkin, et al., 1980; Peck & John, 1992; Young & O'Shea, 1981).

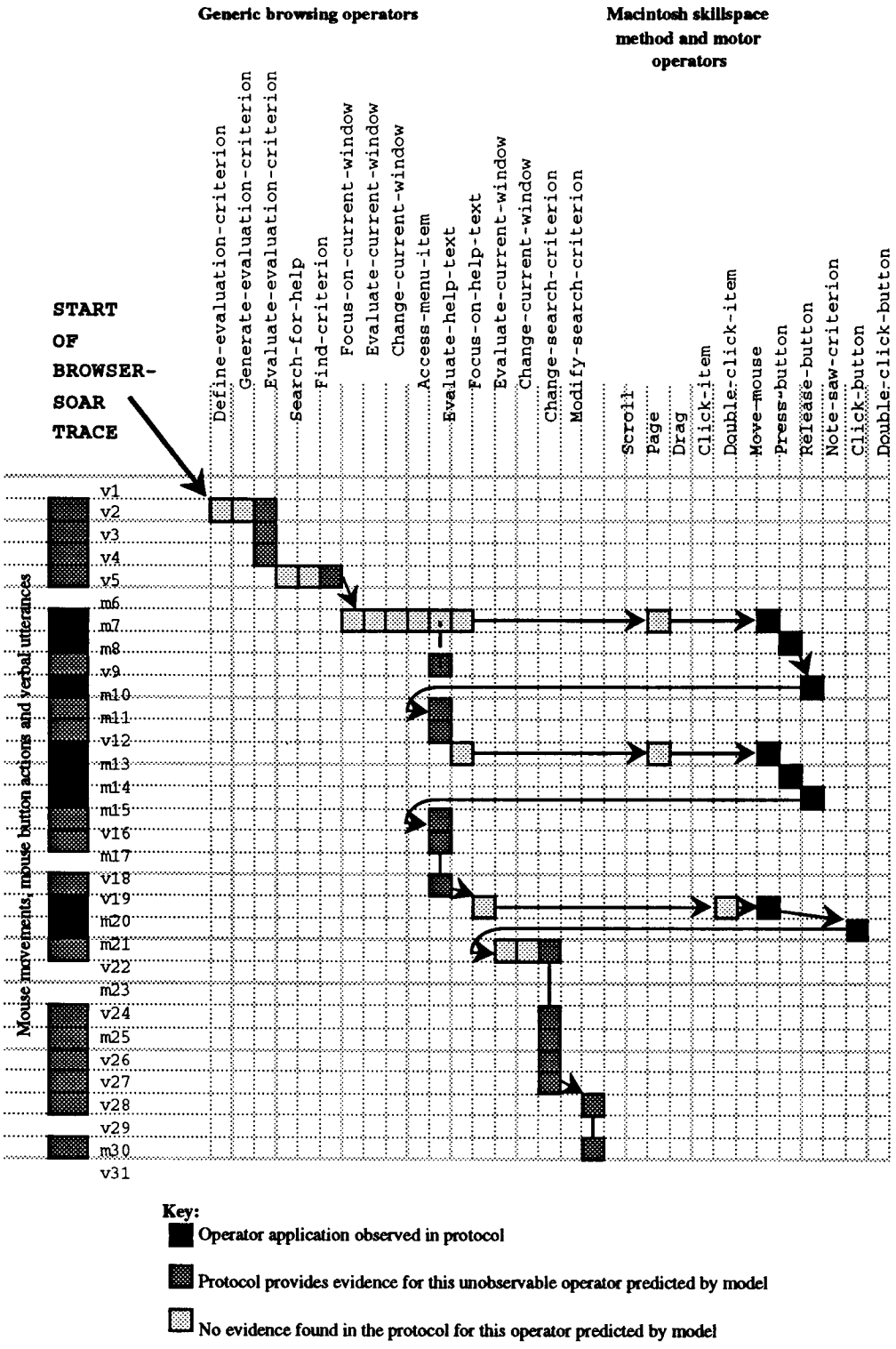

Figure 4. Representation of fit between observed behavior during the "write" browsing behavior excerpt (see transcription in Figure 2) and behavior predicted by the Browser-Soar model

Figure 2-7: Example operator application support graph, from Peck and John, 1992.

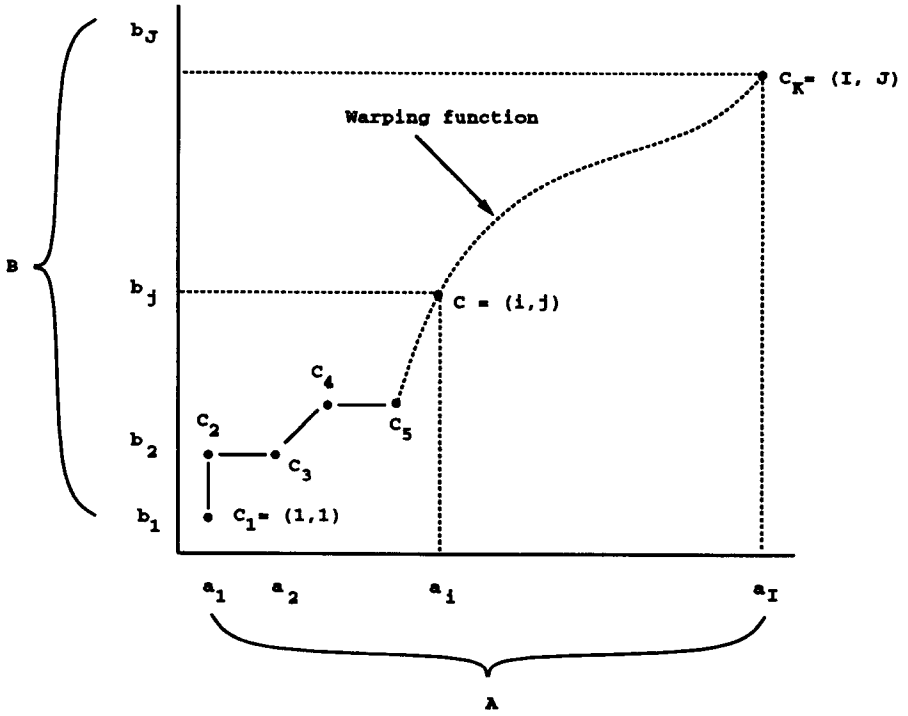

Figure 2-8: Example match over time display, taken from Sakoe and Chiba (1978).

More complex measures have been built from these building blocks in an attempt to combine them into a better summary measure. These measures include the error fit measure (fit = (Hits - FA)/(number of data points)) (Priest & Young, 1988; Young & O'Shea, 1981); and the micro-theory evaluation measure (u = (Hits - number of variant procedures)/(number of data points)) (Priest & Young, 1988). The raw numbers could also be combined in any two-way table of association that allows a missing cell, such as Signal Detection Theory (Swets, 1973; Swets, 1986) (more complete lists have been put together by Nelson (1984) and Reynolds, (1984)).

Simple aggregates and combinations of the match types are easy to compute. They often serve as useful summaries of the fit, and may be persuasive because they can be directly interpreted. If a general evaluation of the model is desired, all the component measures should be reported. Combining them requires additional assumptions about what the measure will be used for, and the relative costs of each type. Combinations cannot serve as measures for improving the model because they do not indicate where to improve the model, only a global measure of its correspondence. The raw components of these measures tell the analyst more about where to improve the model than their combination.

Any of the simple measure of the total numbers and percentages of hits, misses, and false alarms are useful measures for an analyst, it does not appear to matter which one is used, as long as it is readily available and helps indicate where to improve the model. By aiding others in understanding the global quality of the fit, they can also add to the persuasiveness, and should be reported when available. They must be augmented with other measures because on their own they cannot describe where to improve the model or predict how it will do in the future.

2.4.5 Measures of component utility

Measures of rule or component utility are used for measuring how much each additional component of the model adds to the model's performance. These measures are particularly designed to answer the question of the marginal utility of the last piece added. Did adding the last rule make the model fit much better or is it over-fitting the data, picking up just one action? This measure tells the analyst when to stop improving the model based solely on the current set of data.

For process models based on production systems, the most natural measure of the degrees of freedom in the model is the number of rules. In the extreme, it takes a single rule to do each action, and rules could be made for each data point. If fewer rules are required, the model would have less degrees of freedom. Nearly all of the measures included have used rules as their unit of model size, so we will just speak about rules. The only other apparent possibility is number of clauses, which would correlate with the number of rules, and their specificity: more specific rules would have more clauses. Different people may implement essentially the same model using different numbers of productions. These productions will also vary in generality. The number of productions has to be taken then as an upper bound on the degrees of freedom used (i.e., one might be able to create the model with less rules, but need not write more).

Simple counts. The raw number of firings per rule or instantiations per operator, without comparison to the data, is also worth using. As the number of applications per construct goes up, the importance of the rule presumably increases (e.g., Newell & Simon, 1972, Figure 7.29). As a measure of the model alone, it will help the analyst understand the model for later modification, and pointing out rules that have not fired and are perhaps incorrect. It can also be considered as a possible degrees of freedom measure.

Jackknife measures. The most difficult approach is to do a jackknife analysis based on the rules. In this analysis, each rule is taken out individually, the model is run and matched to the data, and a goodness of fit measure is computed, such as number of subject actions matched. By averaging the result and computing the sample variance, the difference in percentage of fit for each rule could be translated into the amount of variance that it accounts for. The significance of any rule could be tested with a plain F-test. Doing this for a process model with hundreds of rules (e.g., Browser-Soar) would be prohibitively expensive, even with tools to automatically do the task. A variant has been used on smaller models generating fixed responses, and they help illustrate the contribution of each sub-model (Young & O'Shea, 1981).

The cumulative hit curve. The cumulative hit curve (e.g., Newell & Simon, 1972, Figure 6.10; Ohlsson, 1980, Figure 6:8; Priest & Young, 1988) is a more specialized graph than a listing of how many times each model component was matched by data. It too depicts the number of data segments accounted for by each rule (or sets of rules), but the rules are sorted in decreasing order of data covered. Figure 2-9a depicts an example diagram. The curve will only reach 100% if all of the data are covered. Generating this curve simply requires keeping a list of rules and being able to aggregate the number of times they are supported by the data. It visually depicts the contribution each rule makes to fitting the data.

The diagram could be made better by more directly showing the measure this graph is actually designed to show, the incremental rule utility. In Figure 2-9a, computing the contribution of a rule requires a relative judgement between two unknown lines. Figure 2-9b shows a graph that shows the incremental amounts as an absolute measurement between a known line (the baseline) and top of a second bar for each rule.

Summary. Providing rule (or operator) counts, firing rates and the frequency of their support in the protocols appear to be useful measures. In addition to providing measures of the utility of the rules, the measures of fit can also serve as debugging aids for process models; rules that don't fire at all, or that fire all the time, are probably wrong in some way and need to be changed. A graphical display of what they match in the protocol is a way of visually displaying this information in a clearer form.

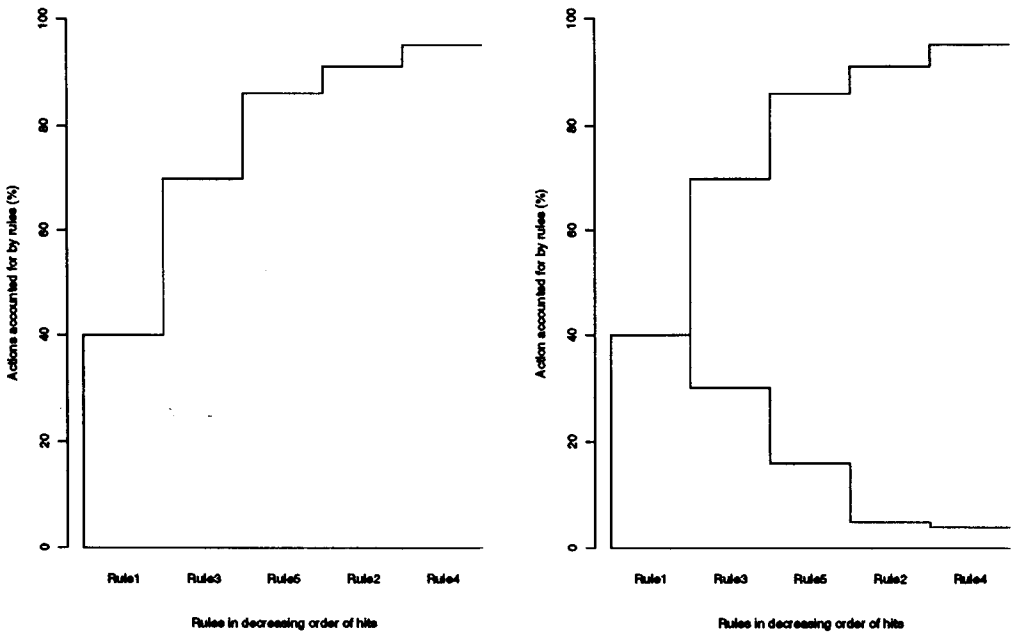

Figure 2-9: (a) Example cumulative hit curve (right). (b) Redesigned cumulative hit curve (left).

These measures have much in common with model based visual displays of the match, and summaries might better start there.

2.4.6 Inferential measures

A common form of persuasion in experimental psychology is reporting for effects the significance levels from inferential statistics. So researchers often looked there first for measures of the quality of model fit. These measures were designed to be a simple test of simple models, for example, that the numbers making up two means were sampled from different distributions. As process models are extreme hypotheses, inferential statistics do not apply. However, the measures that are computed for use in inferential statistics may prove helpful for highlighting where the model could be improved. There is also a general inferential statistic, the variance accounted for, that we can use to make predictions of future performance given multiple subjects.

Is the match better than chance? Examining the general questions of statistical significance first will simplify the remaining discussion. Can we tell if this model is significant?, that is, does it fit better than one might predict would happen by chance? Process models make a large number of predictions, that the multiple sequential responses of the model will match the multiple responses in the data. We are thus testing an extreme hypothesis. What chance is, in this case, is an impossibly rare event. The model will always (or nearly always) match many more tokens in the token sequence than chance would predict (Grant, 1962; Gregg & Simon, 1967). On the other hand, the models will often make predictions that turn out to be wrong, indicating that the model and data are distinctly different, and making the model's probability of being correct be zero. When sampling from different, overlapping distributions this result is not possible. So the answer to this question tells us little, and we must reject the measures of the probability of a model being correct. We can however, increase our belief that the model is more or less correct (the probability of the model) after it has been applied to more data, and the ability to move this informal belief, being persuasion, is one of the criteria for evaluating measures.

Do the model and subject appear not different? Perhaps the question of statistical significance can be redeemed by posing the opposite statistical question of does the model appear to be different from the subject. The two information streams could be compared to see if they are different, and an inferential measure used to determine if the difference is significant. We must dismiss this question too as irrelevant. Failing to reject a hypothesis is a weak claim, and is not a proof of similarity. The model will always (or nearly always will) be different than the behavior in some sense. This will be particularly true given a good data and large amounts of it. Poor experiments with small amounts of noisy data will be unfairly rewarded (this is why the data quality measure was first introduced). Most importantly though, the probability measures provided by the inferential measure does not tell us where to improve the model, nor do they tell how well the model will perform in the future. While we will not find these tests in themselves a useful measure, some of them may highlight useful things about the comparison in any case. The only use they have, is noting that the model is close to performing like the subject (but how close cannot be told).

Frequencies and transition rates. Two measures have been used to test the frequencies of action types and their transition rates. The Chi-squared measure has been misused in at least two different ways as an inferential statistic, but it could provide useful information. The most natural way to misapply it is to test whether the aggregate measures of the model's action types are different than the subject's. This would be an attempt to show that they are do not use the operators in different amounts (or similarly, a different number of items, using an F-test (Karat, 1968)). By accepting the null hypothesis it attempts to prove that they are the same. This measure has also been used to prove that the distribution of action types for a process model "could not have occurred by chance" (e.g., Koedinger & Anderson, 1990, p. 530). The second test makes more sense statistically, but doesn't tell us anything about the model, presumably, the model would not randomly apply its actions, but that is the point of a process model as opposed to a Markov model.

While it fails to be able to prove the model correct, the Chi-squared measure can be used to improve it. The analyst can examine the cell differences between subject and model, choosing to pay attention to the action types that are used in different proportions. This is not a complete measure, but it provides a summary, which is always helpful.

Sequential lag analysis (SLA) (Gottman & Roy, 1990) is also often used as a summary statistic that is useful for building process models as a way to find the major transitions or behavior loops. It can, however, be correctly used in a way similar to Chi-squared. It can note where the transition rates are different between the model and subject, and suggest areas of sequential behavior of the model to improve. It could be misused in ways similar to Chi-squared to test and improve process theories.

Variance accounted for. An inferential measure that can be appropriately applied is testing the significance of the correlation between the subject's actions and the model's actions (Grant, 1962). Tests are available for computing the significance of most types of correlations. Testing correlation rewards good theories by making their fit be significant. Significance is positively related to model quality, data quality, and data quantity. The partial measures contributing to the correlation that are low will indicate where the model could be improved. They will not indicate individual actions, but a category type, providing a global indication of where to improve the model. This is sometimes useful. As correlation can also be expressed as the variance accounted for by the theory, it makes predictions of how well future models will fit.

There are multiple features of a model that can be tested to see that they correlate with the subject's behaviors. Significance tests have been applied to correlations of reaction times on individual item tasks (Card, et al., 1983; John, 1988), to incremental reaction times (e.g., Just & Thibadeau, 1984), and to correlations of state transitions (e.g., Simon & Reed, 1976). There are special regressions available to account for the dependency of sequential measurements (Kadane, Larkin, & Mayer, 1981; Larkin, Mayer, & Kadane, 1986). Cautious modelers might test multiple facets of the model.

Individual subject and model differences. Given multiple subjects or multiple episodes per subject, standard statistical tests can be properly applied to the distribution of measures of fit are being sampled

from a presumably normal distribution. These measures would have to be numeric, and would include the variance accounted for and percentage of the subject's actions predicted, Standard tests (e.g., T-tests) are available to tell if a subject or model was matched significantly more poorly than others. Doing this would be useful to indicate which data set to pay attention to first, but not where in that data set. Visual inspection of the data might also be enough to tell which subject to start with.

In cases where the model is pliable enough to be over-fit, it may be useful to fit the model to one-half of the subjects, and then compare the model's fit to the second half of the data. Non-numeric measures would point out the differences in behavior and in fit between the two halves. In addition to telling more about the model, it may be especially persuasive to other scientists, who could recognize this schema from linear regression.

The differences in fit between two models could be tested in a statistical sense. The test results cannot be accepted as directly as they are when testing linear models however. The question of parsimony and other constraints on the models (if they differ) could greatly influence the final choice. How to weight the correspondence of a model to aggregate data like "human memory shows priming" is unclear.

Summary. Some inferential statistics have interesting submeasures that could be used to highlight where the match could be improved. But they must be used with caution. Checking to see if the model does not appear to be different from the subject may be useful to the analyst as an initial sanity check. It is not even a condition that all models must or can meet. In reporting this number as a result, it is accepting the null hypothesis, which is clearly wrong.

The model can be checked for significance with inferential statistics by testing the correlation between the subject's and the model's performance, through such measures as processing time, state selection, and strategy choices. As a stamp of approval, it is a weak test, most models of some merit will pass it. It is a good measure however, because it both provides help in finding where to improve the model, and given several subjects and a numeric measure of fit, they can give you estimates of how well, in numeric terms, the model will fit in the future. It will also be seen as persuasive.

2.4.7 A unified view: Criterion based model evaluation

A unified approach is also possible towards testing process models. What is actually desired is a model that is consistent with all that is known about human behavior, not merely the results of a small task, or a single subject's actual performance (Newell, 1991). The analyst and their audience need to know how well the model fits the data and what data it fits so that the model can be improved by adding more regularities and so that it can be compared with competing models, including variants of itself. When testing process models this suggests that in addition to fitting sequential data, it can be further constrained by having the model match aggregate data from other experiments. I will label this listing of the regularities accounted for as criterion-based model evaluation. In addition to matching the sequential predictions of a model to data, criterion-based model evaluation compares the model's performance with additional aggregate measures that would also apply to the task being modeled.

In criterion based model evaluation the data regularities and the model's correspondence are laid out in a listing or table. Figure 2-10 shows the type of format of regularities and the model's match for an example set of regularities modeled by FOKIBOFIT-Soar (Ritter, 1989) in the area of feeling of knowing for arithmetic problems. All kinds of regularities are possibilities, nominal performance, error rates and types, and of course the sequential behavior of particular subjects. Table 2-7 shows the five types of regularities that models have been asked to account for. The larger the number of regularities, the better. Various levels of fit, such as qualitative, quantitative, and not-at-all are allowed to describe the quality of fit. How well the model fits each regularity is included in the discussion of the quality of the model.

This approach is not new, but is merely being noted as an actual testing method that is particularly useful. Soar modelers often start with a list of criterion to model, and provide a scorecard of their

model's performance. For example, John (1988) provided a list of 29 regularities modeled by her typing model, Polk (1992) provides a list of 14 regularities accounted for by his syllogism model, and Newell's (1990) book is filled with lists of regularities that a unified theory of cognition must account for (e.g., Figures 1-7 and 5-14).

The FOK 8 — Model #6

Level of Qualitative Match	
XXXX	1. Strategy choice accuracy is pretty good.
XXXX	2. Only operator affects choice time.
X	3. Power law learning for answer time.
X	4. Power law strategy change.
XXXX	5. Linear relationship between learning and strategy choice.
XXXX	6. Frequency of the co-occurrence of the operands best predictor of strategy choice.
---	7. Abrupt changes in multiplication times.
---	8. Retrieving an answer does not guarantee later retrieval.

Figure 2-10: Example criterion table taken from Ritter (1989)

Rarely are lists of criterion used outside of the Soar community (for a counter example see Feldman, Tonge, & Kanter, 1963). For example, the development of Act★ (Anderson, 1983) has such lists implicitly in it, but no where does it include an explicit list of regularities that it covers, or of problem areas remaining to be addressed. For example, none of the 17 papers in the twelfth volume of Cognitive Science (1988) provide a list of regularities covered by their model. More typical tests of a model is to fit a single curve, a single data point, or even to informally exhibit similar behavior (Kaplan, 1987). Why this discrepancy? Are other scientists modeling simpler things? Are they doing it less well? Are the Soar researchers pedantic or irrelevant, or not communicating their methodology and more complex models? It appears that the Soar researchers are headed after more (but certainly not complete) unified theories (Newell, 1990), and with more data laid out in front of them need better ways to keep track of their fit to the multiple regularities. As other psychologists attempt to create larger models, they too will go towards more explicit listing of the regularities covered by their model.

2.4.8 Summary of measures

There are two basic results that the measure of model fit must provide. They must show where to improve the model and they must help predict how well the model will do in the future. There does not appear to be a single measure that will meet both criteria. Models can need improvement in lots of ways and make various types of predictions, so having several measures to do these tasks is acceptable and appropriate.

The analyst appears to end up needing three types of measures. First, a simple local measure to guide the improvement. While not providing the measure, signal detection provides useful terms for creating it. Some weighting of the relative importance of doing what the model does (hits and misses), while not doing what the model does not do (false alarms) is necessary. The degrees of freedom in the model, the number of rules or model resets, should also be kept in mind. These two measures should

Table 2-7: Types of data that have been used to test process models.

1. Sufficiency tests. The model is checked to see that it can perform the task in question in terms of the information it processes and its results, that is, be a functional model (e.g., the Logic Theorist (Newell et al., 1958) creating logic proofs).
2. General behaviors The general types of behaviors can be listed as regularities required in a model. These can include knowledge representations (e.g., visual schemas in geometry, Koedinger & Anderson, 1990), strategies (e.g, scientific discovery, Qin & Simon, 1990), operators and the problem spaces that they work in (the Logic Theorist: Newell et al., 1958). Learning and developmental effects are also powerful constraints (Anzi & Simon, 1979; Newell & Rosenbloom, 1981; Shrager, Hogg & Huberman, 1988; Simon, et al., 1991).
3. Qualitative effects of task stimuli The regularities can provide rather weak requirements for a model to meet. That different sets of problems be found hard or easy (Polk, 1992), and that dependent measures take on different values based on the task stimuli (e.g., typing non-words is not as fast as words, John, 1988)).
4. Quantitative behaviors (can be times or counts) The regularities can be that the model performs tasks at a set speed, or that task must be performed not only at relatively different speeds, but that the relationship in time be a given ratio (John, 1988). Similar constraints are available in relative response rates as well (e.g., Polk, 1992).
5. Sequential action correspondence The perhaps the strongest constraint available is that of sequential behavior — that the model performs a set of actions in a set order (e.g., Peck & John, 1992; Newell & Simon, 1972; Larkin, 1981). Currently, this constraint has only been taken to predict the order, but it could easily be extended to include time between predicted steps as well.

be directly combined. If the subject is being tested and not the model, then some measure combining the relative costs of hits, misses and false alarms is all that is required.

Second, for larger data sets, diagrams and graphs would be useful addition. They are often very compelling ways to display large amounts of information (Larkin & Simon, 1987; Tufte, 1990). They present more information than a single number, and can provide a more global description of the model's performance with respect to the data, helping to find where to improve the model. No visual cliches are available yet to describe the fit of process models' predictions, but there is no reason to believe that they are impossible to obtain.

Third, the analyst needs a measure of how well the model will fare on future data sets. The form this question has often taken is to prove that the model performed better than chance. Process models are extreme hypotheses, and there are no inferential measures for proving them. The only answer is that we end up like all other science, with arguments and evidence and no neat statistical proof. For example, we cannot "prove" special relativity with statistics, just provide arguments for it (which may include the odds of certain measures appearing by chance, but these are just part of the argument, not the argument itself), and show how well it fits the data (Jefferys & Berger, 1992). We can however, test how well the model's performance correlates with the data. This measure's virtue does not lie in providing a stamp of approval — it is a very weak test, that most models will pass. Its virtue lies in that it predicts how well the model will perform in the future, and by examining its constituent parts, it tells where the model could be improved.

The answer to being persuasive now appears to be simple. The model's strengths and weaknesses can

be explained to others with what is used to study and improve the model by their author. This approach may be convincing because of its disarming honesty, but if the measures do work as designed to tell the modeler where the model fits, doesn't fit, and how will it will fit in the future, they should equally well communicate these features to others. Finally, it is worth listing all the data regularities that the model accounts for.

2.5 Previous models of process model testing

This section reviews previous methods for testing process models as put forward in method sections of papers and books, or as implemented as computer programs. These have often been presented as methods for using process models as a way for doing protocol analysis, but are in reality ways of testing process models with protocol data.

The methodology presented in Newell and Simon (1972). Newell and Simon (1972; Newell, 1968) presented a method that has often been misclassified as merely protocol analysis. Its actual goals were to create and test rule-based process theories. Table 2-7 notes the steps in their testing methodology. Ohlsson (1990) presents model-based trace analysis as an interpretation and more explicit description of this methodology. Ohlsson's interpretation emphasizes the subject's behavior as a path through the problem space.

In this methodology, the protocols (verbal and non-verbal) are first summarized into states in a problem-behavior graph (PBG). An initial problem space for solving the task is created based on the task description alone. The rules making up the problem space are compared with the PBGs by hand. Tallies are kept of such measures as when the rules should have fired and when they did. This methodology (e.g., Newell & Simon 1972, p. 165) did not use timing data except for the order of the segments.

Table 2-8: Steps in protocol analysis method (Newell, 1968).

1. Divide the protocol into phrases.
2. Construct a problem space.
3. Plot the Problem Behavior Graph (PBG).
4. Create a production system.
5. Conjecture individual productions.
6. Consolidate the production system.
7. Plot the production system against the PBG.
8. Determine a conflict resolution rule.

Newell and Simon's methodology is incomplete, and does not focus on creating a running program and testing it. The emphasis of their methodology is on "improving the technology for developing theory, rather than for validating theory." (Newell, 1968, p. 183). Although Newell and Simon do not emphasize the routine aspects of testing process theories and protocol analysis, it is inherent in their work. They often use multiple subjects and multiple episodes. But it also takes huge amounts of time (Simon or Newell comment somewhere?). This methodology, based on detailed comparisons by hand, is not realistically set up for large amounts of data and more complicated models.

The theory implemented in Pas-I and Pas-II. Pas-I (Waterman & Newell, 1971) and Pas-II (Waterman, 1973; Waterman & Newell, 1973) specified a theory of building and testing process models as

software systems for automatically performing protocol analysis. The analyst created a series of rule sets that would rewrite the verbal protocol into codes, aggregate the codes into problem behavior graphs, and then these graphs could be displayed, or used to test process theories.

Pas-II supported many of the requirements to be noted in the next chapter. It attempted to automate the complete process, from segmentation through model building. It was designed for routine use, and emphasized automating the analysis. It supported the idea of semi-automatic analysis. But Pas-II failed to support several other requirements that can now be pointed out as important. The reasons for its lack of success, as noted earlier in the review, were equally related to its implementation and testing methodology. The simple parser could probably not provide automatic parsing if it was applied to additional domains. It did not closely incorporate the model and its constructs; this required the analyst to enter similar analysis rules in several steps. It lacked a visual interface and displays are now seen as central to this process. The numerous steps proved tedious. The proposed alignment algorithm between the predictions of a production system and the PBG was based on an incremental and continuous match with backtracking when the correspondence was provably wrong.

Competitive argumentation. VanLehn, Brown, & Greeno (1984) put forth a technique for testing and presenting computation theories of cognition that directly applies to process models. The technique primarily consists of noting how the fundamental principles underlying a model (and not just the implementation details) account for the data, and how similar principles would fail to account for the data. By presenting the principles in this manner, it may be possible to note which model components are necessary. They also call for the establishment of critical facts for cognition, facts that models must account for in order to be considered. The set of these critical facts can be used to compare different variants of a theory, showing why the best one is the one that is necessary because it accounts for the largest number of critical facts. This approach is consonant with the idea that unified theories of cognition should play the "anything you can do I can do better" game (Newell, 1990, p. 507). They present an example argument in their (1984) paper, and apply it elsewhere (VanLehn, 1983; VanLehn, Jones, & Chi, 1991).

Competitive argumentation consists of six components (Vanlehn, 1989):

1. A learning model.
2. Data from human learning.
3. A comparison of the model's predictions to the data.
4. A set of hypothesis (specifications for the model's performance, such as "Students [as realized by a model] expect a lesson to introduce at most one new "chunk" of procedure").
5. A demonstration that the model generates all and only the predictions allowed by the hypotheses.
6. A set of arguments, one for each hypothesis, that shows why the hypothesis should be in the theory, and what would happen if it were replaced by a competing hypothesis.

These are laudable goals that good summaries of models will provide whether or not the models are presented through competitive argumentation. In order to explain which parts of a model accounted for the data, the model must be understandable to the analyst, and when data are accounted for, the model components responsible should be noted. In problem solving behavior, it is probable that much of the critical data will be sequential in nature, and until we deal with process models and their predictions routinely, we will not see the critical data as clearly as we need to. In the end this can be seen as higher level presentation technique, not a testing technique for routine use, although gathering this information does test a model. The methods of competitive argumentation are completely worthwhile, but don't directly address how to test the sequential predictions.

ACM and Cirrus. ACM (Langley & Ohlsson, 1989) and Cirrus (Garlick & VanLehn, 1987; VanLehn & Garlick, 1987) start with a problem space of operators and their effects. They use the coded data to create a process model by specifying the application conditions for the operators. The operator application conditions are added based on information measures, so the model is undergoing a type of test as it is built. This testing process is not one that can necessarily be followed by hand, and appears to assume complete coverage of the subject's actions with operators in the problem space. The output is a decision tree of when each operator will be applied. The subject's actions that are covered by operator actions are presumably ignored, although if they are it would be simple enough to flag them as uncovered. It is not clear how they combine the models over multiple episodes or over multiple subjects.

Given a declarative representation of the problem space including an explicit description of the operators and a description of the environment's features, ACM and Cirrus's aggregate the operator application conditions. This is a feature worthy to include in any tool. They show that machine learning techniques can help summarize the subject's performance, in a form that can be turned into useful model components. The application rules that are learned can provide useful summaries; sometimes these rules will be incorrect reflecting a small sample size. The machine learning algorithms can do some of the abduction task of creating a model, but it is not clear where the difficulty lies, in creating the operators, or in defining the constraints on their application.

ASPM. Analysis of symbolic parameter models (ASPM) takes a process model realized as input/output tables, with a given finite set of parameters, and attempts to fit the model by finding the best set of parameters (Polk, 1992). It is possible to test a model against sequential data (as a series of responses), but it is much easier to test a model against single responses. ASPM can test only well developed models. Model fitting is done with known parameters. The input/output tables must be complete; all operator interactions must be noted there. Exhaustive search of all the sets of parameters is avoided by taking advantage of the structures implicit in the operator tables. ASPM assumes that the model is fixed, and it is the parameters that change. For sufficiently developed models, ASPM guarantees the best fit, but no direct indications of where the fit is poor. So it fails to provide a direct way to improve the model through testing.

Summary. With the description of the other methods now complete and with the short description of TBPA in mind from the introduction, I can note a few interesting similarities and differences between it and previous methods for testing process model's sequential predictions.

The major difference between TBPA and model-based trace analysis (Newell & Simon, 1972; Ohlsson, 1990) is that TBPA compares a full and actual trace of the cognitive model with the protocol, not just productions that could apply, and it works on a higher conceptual level than productions. In addition, model-based trace analysis sees the trace more as a way to analyze the protocols; the segments are hand coded to correspond to one or more of the model's operations (this is a type of analysis that we may wish to support as a preliminary or partial analysis). TBPA sees the trace primarily as predictions to be tested against the protocol data. Because TBPA incorporates an architecture (Soar) that can make time based predictions, it can also compare the model's speed with the subject's speed in doing the task, potentially an important source of constraints on models.

Unlike Pas-II (Waterman, 1973; Waterman & Newell, 1973), TBPA does not attempt to be fully automatic. TBPA directly and explicitly includes the process model and a declarative representation of it that can be used to assist and summarize the comparison. Most importantly, the analyst is not expected to modify the analyses by writing additional production systems, but to string together completed tools.

The presentation techniques of competitive argumentation (Vanlehn, 1989), and the model building techniques of ACM (Langley & Ohlsson, 1989) and Cirrus (Kowalski & VanLehn, 1988; VanLehn & Garlick, 1987) are just that, ways to present and build models. They include and indicate some useful functionalities for any environment that manipulates process models, but none that we must require in order to test them.

The earliest methods for testing process theories were implemented by hand. Although they were often used on large data sets, they were not designed for routine use, that is, applying the same model to multiple data sets, or in sufficient detail to automate them. The later, more automatic systems have generally specified a method that can only be applied to well developed models, and ones that have an excruciatingly detailed specification. TBPA is unique in presenting a method for improving an initial weak model to a stronger model through routine testing, which is what Grant (1962) believes is the basic process in science.

2.6 Summary of lessons for process model testing methodology and tools

Based on this survey of the previous uses of protocol data, the tools for manipulating protocols, cognitive process models, and comparing them, and the measures of model fit, we can enumerate several guidelines for a methodology for routinely testing process models with protocol data.

1. Graphic or tabular displays are required. The amount of information studied, generated, and manipulated when dealing with process models and protocols requires that a graphical presentation can be available. The presentation can done in tables or in diagrams. This requirement applies to the model, the model's behavior, both verbal and non-verbal protocol data, the correspondence between the model and data, and the residuals of any measurement.

2. Automate what you can, avoiding known pitfalls. There is a lot of bookkeeping and analysis tasks that are often done by hand in manipulating protocols, models, and their correspondences. Many of these tasks can be automated in a straightforward way, and they should be, such as the alignment of unambiguous actions. There are other tasks which look similar to these that cannot be easily automated. These will require separate research endeavors. Attempts to automate them will derail work on model testing or even general environment building.

   Parsing, that is, attempting to automatically interpret ambiguous data, particularly verbal data, in terms of a model, is perhaps the largest pitfall. Natural language parsing is not a solved problem. Attempts to include it in model testing have failed and taken much effort to do so. Simpler parses are possible though. Eye movements are now routinely translated into attended areas, and menu clicks by definition translate mouse movements into task actions.

3. The environment must be flexible. The environment the analyst works in must be flexible. A structured process and tool set can be presented, but many analyses will not be supported. Simple support for this starts with the ability to add fields to a display, and ends with the ability to create new analyses within the environment through a macro language. When the environment breaks down, the analysis is either not done, or the analyst must move the data to another environment.

4. Keep the original verbal data available. Verbal data contain a lot of information for model building and for model testing. The original data should be kept around for reference even after they has been coded; the model's performance may force them to be reinterpreted. It is a secret weapon — to gain new inspiration, "clear away an evening, and sit down and reread some protocols" (Newell, 1991).

5. Incorporate the model being tested. Analysis environments must explicitly incorporate the model they are building or testing if they wish to specifically test the model. PAW (Fisher, 1987; Fisher, 1991) only includes operator names, so it can only do tests based on operator names and their pattern of occurrences. Pas-II does not explicitly include a model, but allows the user to put pieces of it in various places, so the tests must be created by the user. SAPA includes the model directly, and thus can test the model

directly.

6. Know where the model is wrong so that you can improve it. Grant's (1962) position of scientists as model builders and improvers is persuasive. Improving the model requires knowing where the model is wrong, not if it is significant. Finding out where it is wrong requires generating the model's predictions, bringing them into close alignment with the data, and keeping the model around so that you can find what components are leading to difficulties (and conversely, which components should be left as is), and modifying the appropriate ones. This is basically the approach that is presented as trace based protocol analysis, presented in the next chapter.

Appendix to Chapter 2: Review of the Card model alignment algorithm

In algorithm theory the task of aligning the model's predictions with the subject's actions is equivalent to the longest common subsequence task. In its most general terms it is an NP-complete problem (Garey & Johnson, 1979). For a fixed language (which exists in this case, the types of behavior actions are fixed), or for a fixed number of sequences (which is also the case, there are two, the subject's and the model's behaviors), the problem is solvable in polynomial time (Wagner & Fisher, 1974) and polynomial space (Hirschberg, 1975). The task specification assumes that the globally best match is required, that there are no special correspondences that must be tied together, and that all subject data are used (and there are not any bits discarded as noise). These could, of course, be handled through simple extensions.

How the alignment is produced must be clear. The final alignment must be editable; even if the alignment is done completely and automatically, the analyst may have to jump in and redo parts by hand, either to correct the alignment because the specification of alignable objects was wrong, or to play what-if games. Our main interest in finding this subsequence is to use it to actually align the model's predictions with the appropriate data, rather than as a measure of similarity as it is often used. This gives us slightly different interests and needs. There are several that are simple and direct.

A potential problem is that the maximally common subsequence may not be unique — there may be several possible — and that we may have preferences about which one is returned. If we are just interested in the length of the maximally common subsequence, or the percentage of each sequence matched, which subsequence returned doesn't make any difference. For this task we have two preferences. First, we would like the longest subsequence that starts from the front. Since both the model and the subject move forward in time, the match must begin there. We believe that a priori the model's earlier predictions will be more accurate, and as modelers, we most often and most easily adjust the model's behavior starting with its initial actions. As we iterate through the cycle of match, modify the model, match, the earlier matches will be more stable, and require less modifications over time.

Second, the model's actions should guide the match. It represents the theoretical terms of the task, and in most cases there will be less model actions than subject actions, and these actions are less noisy, for our models don't have additional processes to add noise to their behavior like subjects have. Since the actions must actually match each other, and earlier actions are preferred to be included in the subsequence over later actions that also match, this constraint is also satisfied, although no additional changes will be required to the common implementations of the algorithm.

There are also requirements on the two types of data streams to be aligned. They cannot be ambiguous if the alignment is to be done automatically. This is a known problem of verbal utterances, but can also be found in a model trace if the token "add" is used to refer to two different types of constructs. Partial coding of the verbal data may be desirable here; coding polysemous words to a specific meaning. Having multiple data streams will also help; non-verbal discrete actions surrounding verbal utterances will help constrain the match of the verbal utterances.

The algorithm presented by Card, Moran, and Newell implements this algorithm by computing a matrix of possible matches, and then walks through this matrix generating the longest subsequence that matches. The initial step creates a matrix of counters, called SCORE, of size NUMBER-OF-OBSERVED by NUMBER-OF-PREDICTED. It then compares in order each token in the observed sequence against each token in the predicted sequence. Where there are matches, the counters are incremented to represent the longest possible sequence starting from Matrix(Observed-token, predicted-token). The final value of SCORE (SCORE[NUMBER-OF-OBSERVED, NUMBER-OF-PREDICTED]) is the size of the longest possible common subsequence. The second and final step traverses the matrix backward, starting at its most extreme point, generating one of the possible maximum length subsequences. Consider the example in Figure 11 matching DUC and DUDUDU.

```
Matching   predicted: D U C
and        observed:  D U D U D U
would generate the score matrix:

           D U D U D U
         0 1 2 3 4 5 6

     0   0 0 0 0 0 0 0
          \
   D 1   0 1 1 1 1 1 1
            \
   U 2   0 1 2 2 2 2 2
              |
   C 3   0 1 2-2-2-2-2

Alignment returned by Card1:

   Predicted sequence:       - - - - C U D
   Observed sequence:        U D U D - U D
```

Figure 11: Example alignment by the Card1 algorithm. The two strings being aligned are "DUC" and "DUDUDU".

How well the match was performed, the percentage matched, may be scored with Card's formula

Length(common-subsequence) / Max(length(observed), length(predicted)) (1)

This may be a bit pessimistic, in that it assumes that the longer sequence should be completely matched. An alternative formula 2 divides by the length of the shorter sequence, telling how many actions were matched that could be matched, taking the sequences as givens. The subsequences will be used for editing, so each of these measures are only used to inform the analyst how much editing by hand may remain to be done.

Length(common-subsequence) / Min(length(observed), length(predicted)) (2)