Behavior Research Methods, Instruments, & Computers
2001, 33 (2), 149-158

ACT-PRO action protocol analyzer: A tool for analyzing discrete action protocols

WAI-TAT FU
George Mason University, Fairfax, Virginia

This article presents a top-down approach for analyzing sequential events in behavioral data. Analysis of behavioral sequential data often entails identifying patterns specified by the researchers. Algorithms were developed and applied to analyze a kind of behavioral data, called *discrete action protocol data*. Discrete action protocols consist of discrete user actions, such as mouse clicks and keypresses. Unfortunately, the process of analyzing the huge volume of actions (typically, $>10^5$) is very labor intensive. To facilitate this process, we developed an action protocol analyzer (ACT-PRO) that provides two levels of pattern matching. Level one uses formal grammars to identify sequential patterns. Level two matches these patterns to a hierarchical structure. ACT-PRO can be used to determine how well data fit the patterns specified by an experimenter. Complementarily, it can be used to focus an experimenter's attention on data that do not fit the prespecified patterns.

This paper presents a software tool that automates the analysis of sequential behavioral data. The tool was applied to the analysis of a specific kind of behavioral data, called *discrete action protocols*. Discrete action protocol data are collected as subjects are working on a task in a computer system. A flow of high-resolution time-stamped events can be easily collected from the application, the input, and the output devices being used. For example, in a study of the Tower of Hanoi (see, e.g., Simon, 1975), which uses a direct manipulation interface in which disks are dragged from one pole to another, discrete action protocols might include a record of each disk clicked, the pole from which it originated, and the pole to which it was moved. Each discrete action—mouse down at one location and mouse up at another—would be time stamped and saved to a log file. Each action can be unambiguously defined as a discrete event, and each set of discrete action protocols collected from subjects is a set of linear sequential events. As compared with verbal protocol data, discrete action protocols are more constrained and easier to interpret, and data from multiple subjects are easier to aggregate. However, extensive analysis of a large volume of discrete action protocol data is time consuming and error prone. The building of a tool to analyze discrete action protocols automatically not only can reduce the effect required, but also can increase objectivity by using the same set of determinate rules throughout the entire set of protocols.

The tool developed is called ACT-PRO (which stands for action protocol analyzer). It provides a top-down method to automate the analyses and can be used to facilitate the iterative process of theory construction. ACT-PRO automates the process of matching a huge amount of data to a researchers' theory, and the output trace from ACT-PRO can highlight which parts of the sequential data match and do not match the theory. This kind of information is very useful for researchers to extend and refine their theory. However, all analyses of sequential events have much in common, whether the sequential events are obtained from action protocols or from more complex, naturalistic event sequences. Hence, a review of techniques for analyzing naturalistic sequential events will be presented and will be followed by a comparison of these techniques to ACT-PRO. I will then focus on how ACT-PRO can be used to analyze action protocol data and will provide examples of applying ACT-PRO.

Finding Patterns in Sequential Events

Existing methods for finding patterns in sequential events are either exploratory or confirmatory. Exploratory methods are used in situations in which the temporal structure of behavior is largely unknown. On the other hand, confirmatory approaches involve matching the theoretical sequential patterns specified by researchers to the observed sequences. Traditional methods adopt general sequential statistical techniques to seek statistical dependencies between events over time. For example, *Markov analysis* and *lag sequential analysis* test the significance of transitions of events and identify chains of events that occur significantly more often than others (e.g., Bakeman & Gottman, 1997). These techniques have been much used, and efforts have been made to develop a standard format for representing sequential data so that these gen-

The present research was supported by Grant IRI-9618833 from the National Science Foundation, as well as by Grant AFOSR#F49620-97-1-0353 from the Air Force Office of Scientific Research. Thanks to Wayne D. Gray for the encouragement and the opportunity to develop ACT-PRO. Thanks to Wayne D. Gray, Michael Schoelles, Christian Schunn, Greg Trafton, Melanie Diez, Tony Harrison, Bill Liles, and Lelyn Saner for feedback on earlier versions of this manuscript. Thanks to Vicenc Quera and an anonymous reviewer for valuable suggestions on improving ACT-PRO and the present paper. Correspondence should be addressed to W.-T. Fu, George Mason University, Psychology, MS 3F5, Fairfax, VA 22030 (e-mail: wfu@gmu.edu).

eral sequential analysis algorithms can be applied (e.g., Bakeman & Quera, 1992, 1995).

The main difficulty in finding patterns in sequential events is that researchers are seldom interested in identifying a rigid chain of events but are rather interested in a family of sequential events that share a common underlying structural pattern. To solve this problem, researchers have come up with different ways to measure *similarity* between sequences to obtain quantitative measures of the distance between sequences (for a good review of these measures, see Sankoff & Kruskal, 1983). One common distance measure between two sequences is the smallest number of *elementary edit operations*, such as *substitutions*, *insertions*, and *deletions*, that transform one sequence into another. For example, to transform the sequence "aabcdd" to sequence "aaabed," one can first *insert* "a," *substitute* "c" for "e," then *delete* "d" (see Table 1). The smallest number of elementary edit operations, and thus the distance between the two sequences, is therefore three. Similar sequences (i.e., sequences that have small distances between them) can be clustered together as representing the same underlying structure that the researchers are interested in.

Recently, domain-specific approaches were developed to capture patterns that are specific to the characteristics of the data. These approaches assume that specific forms of patterns exist in the data, and corresponding algorithms are developed to look for these patterns. For example, algorithms were developed to identify buying patterns in large databases of customer transactions (Agrawal & Srikant, 1995), surfing patterns on the World-Wide Web (Pitkow & Pirolli, 1999), and recurring behavioral patterns in dyadic interactions (Magnusson, 2000). By exploiting the characteristics of the data, these algorithms are demonstrated to be efficient in capturing patterns that the researchers are interested in.

Overview of ACT-PRO

ACT-PRO performs two levels of pattern matching. Both levels of pattern matching can be applied to the same data set, or either level can be used separately. The first level involves identifying sequences of actions that are grouped or chunked together. The second level matches the data to a hierarchical structure specified by the researcher. In the first level, ACT-PRO uses techniques in formal grammar to capture various patterns in sequential events. It requires the researchers to specify the structural pattern they are looking for in the data. The structural pattern is represented in the form of a grammar. Each of the grammars has a set of grammar rules, with each of the rules roughly corresponding to the elementary edit operations mentioned above. The formal grammar representation therefore provides a natural way of measuring distances between sequences. Multiple sequences that can be transformed into the same sequence (i.e., the same structural pattern specified by the grammar) by the same set of grammar rules will be clustered together and captured by the grammar. Detailed examples will be given in the next section.

Table 1
Example of a Distance Measure Between Two Sequences

Sequence Transformation	Elementary Edit Operations
"aabcdd"	
↓	Insert "a"
"aaabcdd"	
↓	Substitute "c" for "e"
"aaabedd"	
↓	Delete "d"
"aaabed"	

Note—Distance between the two sequences is defined as the number of elementary edit operations that transform one sequence to another. In this example, the distance between "aabcdd" and "aaabed" is three.

Many researchers have argued, either theoretically or empirically, that many kinds of action protocol data can be described by hierarchical structures (Card, Moran, & Newell, 1983; Simon, 1996). In fact, many validations of behavioral theories involve the validation of the hierarchical structures derived from those theories. Therefore, matching a vast amount of action events to the theoretical hierarchical structure is an important step in theory validation. In the second level of pattern matching, ACT-PRO exploits the prominent characteristic of action protocol data and uses a simple algorithm that matches the data to the theoretical hierarchical structure. Details of the algorithm, as well as examples of application, will be discussed in the following sections.

In addition to pattern matching, ACT-PRO also generates simple goodness-of-fit measures that evaluate how good the groupings are and how well the sequences of groups fit the hierarchical structure. These measures are useful for evaluating the researchers' theories. ACT-PRO also produces output traces that highlight parts of data that match and do not match the hierarchy. This allows researchers to expand and modify their theories by focusing their attention on those parts of the data that do not match their theory.

In the following paragraphs, the basic mechanisms of ACT-PRO will be introduced in the context of programming a simple VCR . This example will enable us to illustrate the mechanisms of ACT-PRO without being sidetracked into explaining the details of an unfamiliar technical task. The results of applying ACT-PRO to three different sets of action protocols will be presented. The results will help in evaluating the performance and generality of ACT-PRO. Uses, limitations, and possible extensions of ACT-PRO will be discussed in the last sections.

THE ACTION PROTOCOL ANALYZER—ACT-PRO

Basic Structure

ACT-PRO has two major components: the *grouping program*, which partitions the stream of discrete actions into sets of labeled strings, and the *tracing program*, which maps the action protocols to the hierarchical structure.

Figure 1 shows a hierarchical structure of programming a VCR,[1] which will be used as a running example.

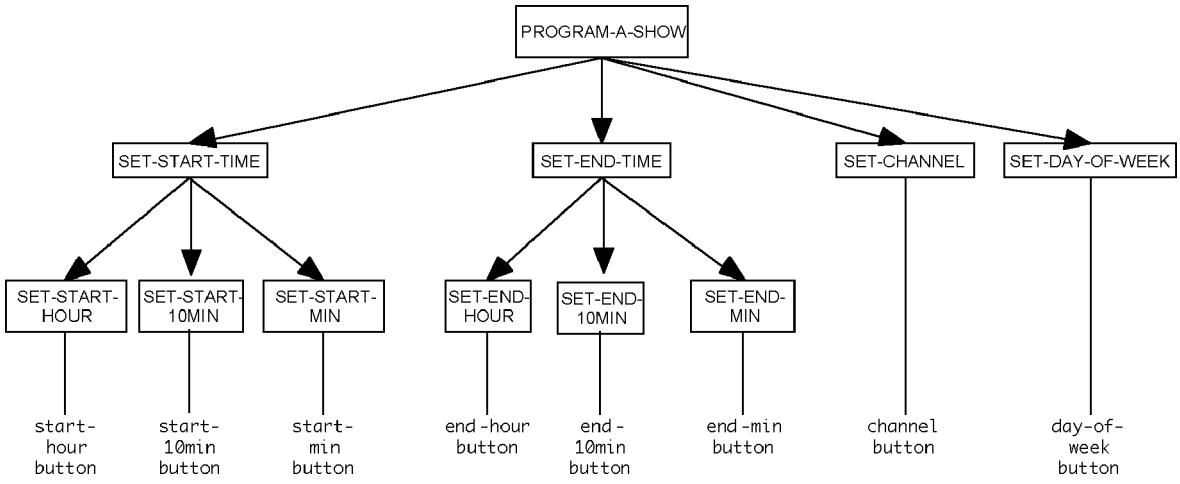

Figure 1. The simplified task-relevant hierarchical goal structure of a VCR interface (Gray, 2000; Gray & Fu, 2000).

In this particular hierarchical structure, the boxed nodes denote the goals, and the unboxed nodes denote the buttons on the VCR that need to be pressed to accomplish the goals.[2] The top-level goal of the task is to program the VCR to record a show. There are four subgoals under the top-level goal: SET-START-TIME, SET-END-TIME, SET-CHANNEL, and SET-DAY-OF-WEEK. To set the start time or the end time, the hour, the 10-minute, and the minute have to be set separately. Notice that the goal hierarchy does not impose any constraints on the order of the subgoals, so long as they are set contiguously. For example, the goals SET-START-TIME, SET-END-TIME, SET-CHANNEL, and SET-DAY-OF-WEEK can be accomplished in any order. Similarly, for SET-START-TIME, the subgoals of SET-START-HOUR, SET-START-10MIN, and SET-START-MIN can be set in any order, so long as they are not separated by extraneous subgoals, such as SET-CHANNEL. The unboxed nodes denote the buttons that need to be pressed to achieve the goals. For example, to set the start hour from 2 to 4, the *start-hour* button needs to be pressed twice, each time incrementing the start hour by one. All other goals can be accomplished in a similar way. Note that, in the simple VCR interface assumed by Figure 1, the procedure for accomplishing each terminal goal is simply the repeated pressing of a single button.

Each labeled string generated by the grouping process maps to one low-level goal (e.g., the three discrete actions—*start-hour*, *start-hour*, *start-hour*—map to the SET-

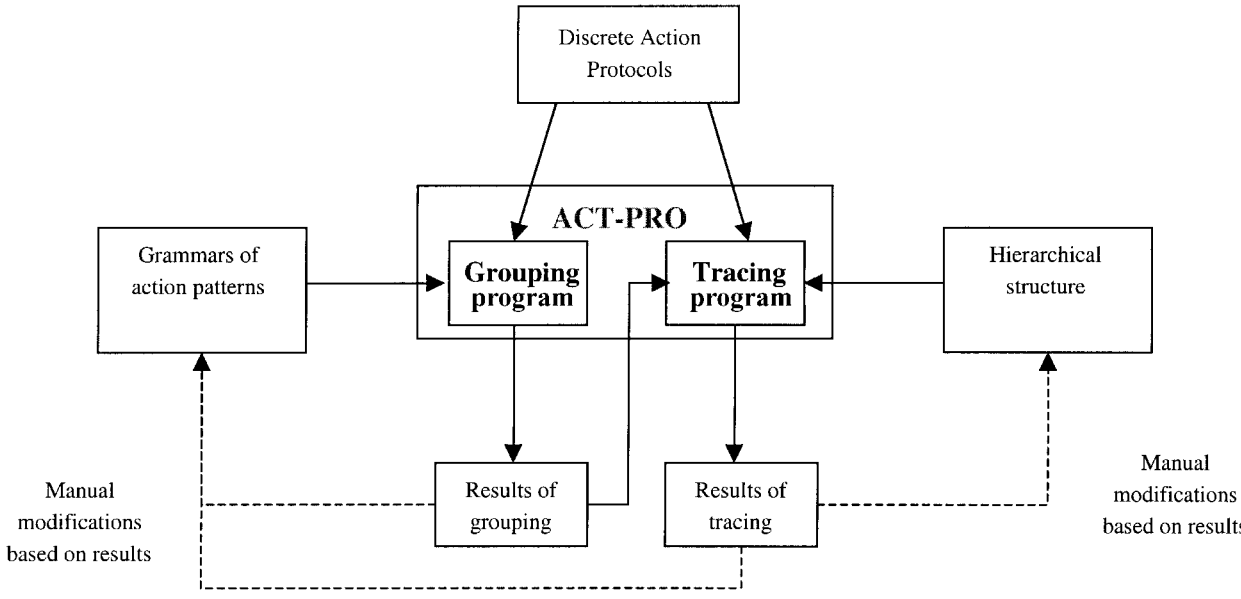

Figure 2. The grouping and tracing programs of ACT-PRO. Solid lines represent direct input; dotted lines represent manual modifications based on the results of the program outputs.

START-HOUR GOAL). In this case, the output of the grouping process is the label SET-START-HOUR and the three discrete actions—*start-hour*, *start-hour*, and *start-hour*.

The tracing program takes both the action protocols and the output of the grouping program as inputs, maps them to the hierarchical goal structure, and outputs a trace of the action protocols with measures of how well they fit the hierarchical structure (see Figure 2).

ACT-PRO requires researchers to specify in advance the procedures or groups of actions that accomplish each task, as well as the hierarchical structure. However, working out the complete set of procedures and the hierarchical structure for even a simple task can be demanding. Automatizing the validation process enables researchers to develop these procedures and the hierarchy iteratively. Researchers can begin with a small set of procedures and a flat hierarchy. Each of these can be modified and expanded after evaluating the outputs of the grouping and tracing programs.[3]

The Grouping Program of ACT-PRO

A syntactic representation of action patterns in a procedure. The main difficulty in the grouping process is that researchers are seldom interested in capturing any single rigid chain of actions. Instead, action sequences that contain the same underlying structures are usually what the researcher is looking for. For example, if a researcher is trying to identify action sequences that accomplish a particular goal, he or she may want to have a single representation that can capture all possible variations of the action sequences that accomplish the same goal. These variations can be true variations—for example, different ways to accomplish the same goals adopted by different individuals—or they can be due to errors or slips or occur for some other reason. The grouping process has to be flexible enough to accommodate the various alternative action sequences that the researcher wants to categorize under the same pattern label. We chose to use the syntactic method to represent the action patterns (for a good review, see Olson, Herbsleb, & Rueter, 1994). Formal grammars are used to capture the variations of the action sequences.

Formal grammar is a general representation that can capture action patterns in various forms. It can capture embedded structures that cannot be captured by most methods. Each grammar has a set of grammar rules; each rule roughly corresponds to the elementary edit operations used in measures of distances between sequences (see Table 1), such as the Levenshtein distance (e.g., see Sankoff & Kruskal, 1983). Therefore, the grammar representation can be viewed as a representation of a prototype sequence pattern, and sequences that can be transformed into this prototype pattern by applying the set of grammar rules in the order defined by the grammar will be clustered and captured. For example, imagine another VCR interface, in which to set the channel from 4 to 7, the subject must press the *channel* button, press an *up-arrow* button three times (each time incrementing the channel by one, hence setting the channel to 7), then press the *enter* button (to confirm the changes made to the channel). However, for various reasons, users may not strictly follow the same action sequence. The researchers may, for example, believe that sometimes the user may press the *channel* button, press the *up-arrow* button four times (setting the channel to 8), press the *enter* button (setting to the wrong channel), press the *down-arrow* button once (setting the channel back to 7), then press the *enter* button again (an error correction). In another situation, the user may press the *channel* button, press the *down-arrow* button once (going in the wrong direction and setting the channel to 3), press the *up-arrow* button four times (setting the channel to 7), then press the *enter* button. To capture all of these variations, a grammar for the task SET-CHANNEL can be written as in Table 2.

Clause 1 defines the grammar SET-CHANNEL as having three objects in the specified order. Clause 2 specifies that [object1] has to be a *channel* buttonpress. Clauses 3–7 specify that [object2] can be any combination of the *up-arrow*, *down-arrow*, and *enter*, except that *enter* cannot be the last buttonpress of [object2]. Clause 8 specifies that [object3] has to be *enter*, which is also the only buttonpress that can terminate the action sequence captured by the grammar. The grammar can flexibly capture many variations of action sequences that match the set of grammar rules (i.e., Clauses 1–8) in the specified order. The syntactic representation provides a flexible mapping between the prototype sequence pattern specified by the set of grammar rules and the vast number of variations of action sequences collected from subjects. A detailed example of how the mapping is done will be given below.

Parsing the action protocols by using the grammars. The action protocols are parsed by using the set of grammars constructed. Starting from the first action in the sequence, the actions are matched to the set of grammars in parallel. Each action is tested against each of the grammar rules defined in each grammar. When a matching grammar is found, the next action is tested against the next grammar rule in the same grammar. When a sequence of actions matches all the grammar rules in a grammar in the specified order, the sequence of actions is "captured" by the grammar. For example, if the sequence of actions channel, up-arrow, down-arrow, enter is matched to the grammar in Table 2, the first action, channel, will first be matched to Clause 2. Since they match, channel will be

Table 2
Grammar That Captures Variations of the Action Sequence Formed by Pressing the Buttons Channel, Up-Arrow, Down-Arrow, and Enter

SET-CHANNEL: [Object1][Object2][Object3]	(1)
[Object1] → *channel*	(2)
[Object2] → *up-arrow*	(3)
[Object2] → *down-arrow*	(4)
[Object2] → *up-arrow* [Object2]	(5)
[Object2] → *down-arrow* [Object2]	(6)
[Object2] → *enter* [Object2]	(7)
[Object3] → *enter*	(8)

taken as [object1] and according to the grammar, the next action has to fit the grammar rules for [object2]. Therefore, the second action, up-arrow, will be matched to Clauses 3–7. This time, both Clause 3 and Clause 5 match the action. Consider the case in which up-arrow is matched to Clause 3; then, up-arrow will be taken as [object2], and the next action will be matched to the grammar rule for [object3] (i.e., Clause 8), which states that [object3] has to be enter. Since the third action is down-arrow, no match is found; the program backtracks and tries to match up-arrow to Clause 5. Clause 5 states that the next action will also be an [object2]; therefore, the third action, down-arrow, will still be matched against Clauses 3–7. This time, down-arrow will match Clause 4, and the next action, enter, will match Clause 8, thus providing a match between the sequence and the grammar as a whole.

It is possible that there is more than one way a stream of actions can be matched to the set of grammars. There is also no guarantee that the grammars constructed are mutually exclusive (i.e., capturing completely different sequences). Some kind of criterion is needed to decide whether one match is better than another. For example, consider the action sequence shown in Figure 3. Each of the three rows shows the same action sequence, parsed by different combinations of the hypothetical grammars G1 to G6. The best-fitting combination can be calculated as the percentage of actions captured by the grammars. In the example shown in Figure 3, the number of actions captured by the combination of G1 and G2 is 9, by G3 and G4 is 10, and by G5 and G6 is 12. Hence, the percentages of actions explained by each of the combinations would be 75%, 83%, and 100%, respectively. During the parsing process, ACT-PRO picks the combination of grammars that can explain the largest number of actions in the action sequence as the output of the grouping program. Therefore, in the example shown in Figure 3, G5 and G6 will be picked as the output of the grouping process.

The Tracing Program of ACT-PRO

The tracing program takes both the action protocols and the output of the grouping program as input and outputs a trace of the action protocols according to the hierarchical structure provided by the researchers. A simple goodness-of-fit measure will also be generated to indicate how well the data match to the hierarchy (details follow). Similar to the specifications of the grammars in the grouping process, the construction of the hierarchical structure may involve an iterative process. After the results of the tracing program are examined, the researchers may manually modify the hierarchical structure. This modified hierarchy can be tested against the action protocols again, and the process can be continued until a satisfactory hierarchy is obtained.

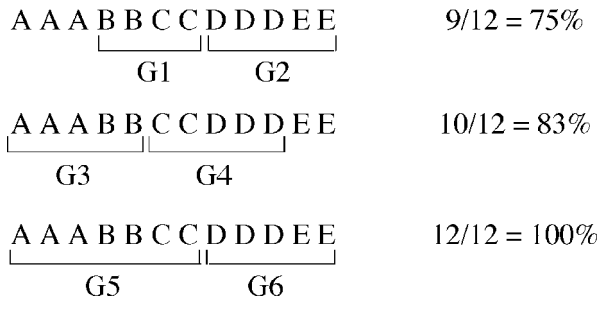

Figure 3. Different combinations of grouping grammars. The percentages indicate the number of actions captured divided by the total number of actions.

Representation of the hierarchical structure. The hierarchical structure is represented by a set of pushdown stacks. The pushdown stack representation preserves the relationship between the layers in the hierarchy and supports a simple mechanism for matching the data to the hierarchy. For example, the hierarchical structure of the VCR programming task shown in Figure 1 can be represented as the set of pushdown stacks in Figure 4. Each of the stacks represents the series of higher level goals that lead to the corresponding action. The set of pushdown stacks can then be used to trace the action protocols by a series of push and pop operations, as will be described below.

Tracing the action protocols by using the hierarchical structure. The tracing program partitions the action protocols and produces a trace according to the hierarchical structure. The tracing process is best illustrated by an example. By using the stacks in Figure 4, the trace shown in Table 3 for the four consecutive actions start-hour, start-hour, start-10min, channel, start-min can be produced as follows. The tracing process starts with an empty pushdown stack. To lead to the execution of pressing the start-hour button, the top-level goal PROGRAM-A-SHOW is pushed on the stack first. Then the goal of SET-START-TIME is pushed on the stack, on top of the goal PROGRAM-A-SHOW, followed by the push of the goal SET-START-HOUR (Figure 5). Each time a goal is pushed to the stack, a trace is produced in the output. Starting from the second action, the tracing program compares the pushdown stack with the series of goals that leads to the second action. Since the second action is also pressing the start-hour button, the series of goals that leads to the second action is the same as that in the pushdown stack. Thus, nothing is pushed or popped. The third action, pressing the start-10min button, requires the pushing of PROGRAM-SHOW, SET-START-TIME, and SET-START-10MIN. To convert the current stack to the one that leads to the action of pressing the start-10min button, the current element at the top of the current stack, SET-START-HOUR, has to be popped (see Figure 6), followed by pushing of the new element, SET-START-10MIN, to the current stack. Similarly, to lead to the fourth action, pressing the channel button, the tracing program pops SET-START-10MIN, pops SET-START-TIME, and pushes SET-CHANNEL. The process continues until the last action is traced; in this case, all the elements in the stack are popped off one by one, with the one on the bottom being popped off last.

Validating the hierarchical goal structure. To validate the hierarchical structure, ACT-PRO classifies each push and pop as either a match or a mismatch to the hierarchical structure. The right column of Table 3 shows

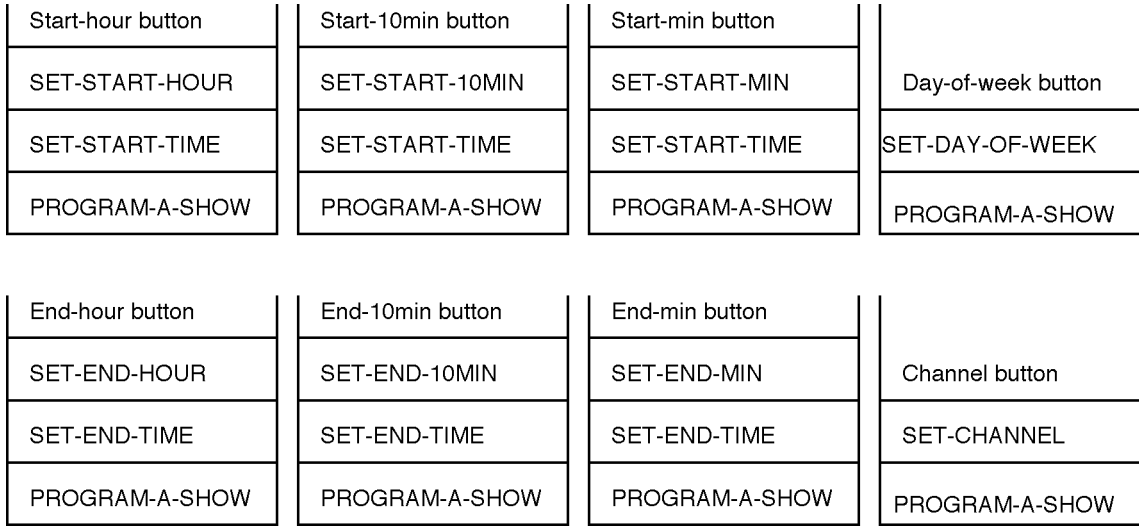

Figure 4. The set of pushdown stacks that represent the hierarchical structure in Figure 1.

the results of the validation process. In the example, the goal SET-START-TIME has to be popped prematurely. This is because, according to the hierarchy, the pattern SET-START-TIME has three components—SET-START-HOUR, SET-START-10MIN, and SET-START-MIN—but now only the first two are found in the data. There is therefore a mismatch between the data and the hierarchy, and the tracing program will flag the popping of SET-START-TIME as a pop mismatch, which is defined as popping an element before it has been completed.

The number of mismatches can be used as a goodness-of-fit measure of how well the hierarchical structure fits to the action protocols. If the match is low (high number of mismatches), the researchers may wish to modify their hierarchical structure (or even the action pattern specifications) and repeat the process until a hierarchy that has an acceptable level of goodness of fit is obtained. The theory of the researchers can be validated by examining how well the data match the hierarchy, and the mismatches can be examined to provide insights into how the theory can be extended and refined.

Table 3
An Example of the Trace and Validation Results of Using the Hierarchical Goal Structure of Figure 1

Trace	Validation results
Push goal: PROGRAM-SHOW	Push goal match
Push goal: SET-START-TIME	Push goal match
Push goal: SET-START-HOUR	Push goal match
Action: *start-hour*	
Action: *start-hour*	
Pop goal: SET-START-HOUR	Pop goal match
Push goal: SET-START-10MIN	Push goal match
Action: *start-10min*	
Pop goal: SET-START-10MIN	Pop goal match
Pop goal: SET-START-TIME	*Pop goal mismatch*
Push goal: SET-CHANNEL	Push goal match
Action: *channel*	
Pop goal: SET-CHANNEL	Pop goal match
Push goal: SET-START-TIME	Push goal match
Push goal: SET-START-MIN	Push goal match
Action: *start-min*	
…	

Hardware and Software Requirements of ACT-PRO

Both the grouping and the tracing programs are written in Macintosh Common Lisp (MCL 4.3). MCL conforms to the common lisp standard, and hence ACT-PRO should be easily transformed to run under an environment that meets this standard. ACT-PRO requires MacOS 7.5 or greater and requires at least 16 MB of memory. The validation program is written in Visual Basics for Excel, which requires Excel 98 running on MacOS 7.5 or greater. The program can be downloaded for free from http://www.hfac.gmu.edu/~wfu/act-pro.htm. Requests for source codes can be sent to wfu@gmu.edu.

EXAMPLES OF APPLYING ACT-PRO

ACT-PRO is applied on three different data sets: the first from a simple VCR interface, the second from a blocks world task, and the third from a perverse-VCR interface (which is very different from the first one). The focus of the present section is on exploring the limits and functionality of the two programs that make up ACT-PRO. (Examples of the use of ACT-PRO in psychological research can be found in Fu & Gray, 2000a, 2000b, and Gray & Fu, 2000.)

The Grouping Program

The uses of the grouping program focus on how the formal grammar representation can capture the variations of action patterns in the procedures. To test the gen-

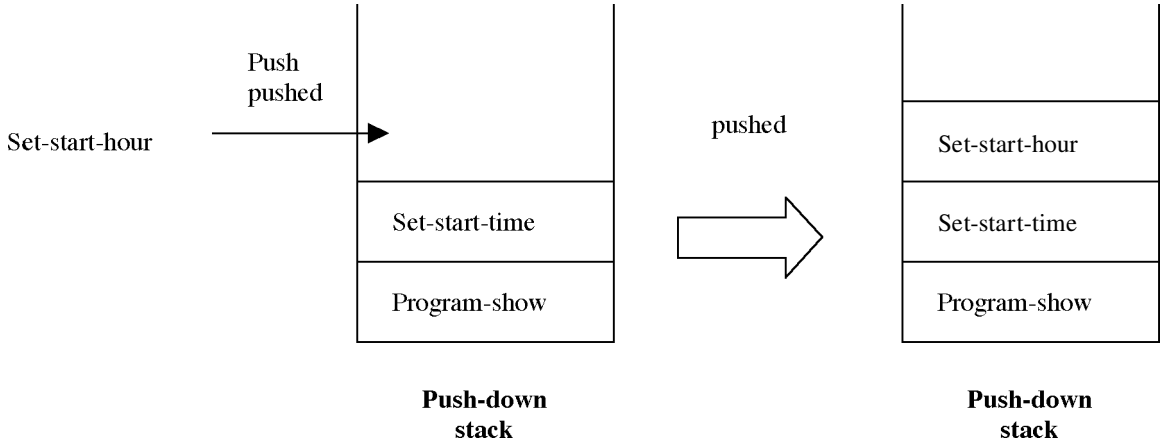

Figure 5. The new element, SET-START-HOUR, is pushed to the pushdown stack. The series of elements pushed to the stack eventually leads to the execution of actions.

erality of the grouping program, we applied the grouping program to two sets of action protocols. The first set of protocols was collected as subjects programmed a perverse-VCR interface on a computer. The second set of protocols was collected as subjects worked on a simple blocks world task (Fu & Gray, 2000b).

Action protocols from a perverse-VCR interface. Action protocols were collected from 64 subjects as they programmed a perverse-VCR interface on a computer. In this interface, the settings are changed by first clicking a radio button that indicates the to-be-changed setting—for example, channel. Then, the subject repeatedly presses the up-arrow or the down-arrow button until the desired setting is reached. Finally, the enter button is pressed to confirm the setting. Each subject programmed eight shows to the criterion of two successive trials. Only trials that ended with the show's being successfully programmed were analyzed. There were 1,228 successful trials, with a total of 51,232 actions (button clicks). Sixteen grammars were constructed for this task, each of them representing a structural pattern. Table 2 shows an example of the grammars used during the grouping process.

The matching calculated the percentage of actions explained by the grammars for each trial. Out of the 1,228 trials to which the grouping program was applied, 81.1% of the worst-fitting trial was matched by the grammar, as was 100% of the best fitting trial. Over all 1,228 trials, the grammars matched the actions for an average fit of 95.1%. The percentages were calculated by the number of actions parsed by the grammars divided by the total

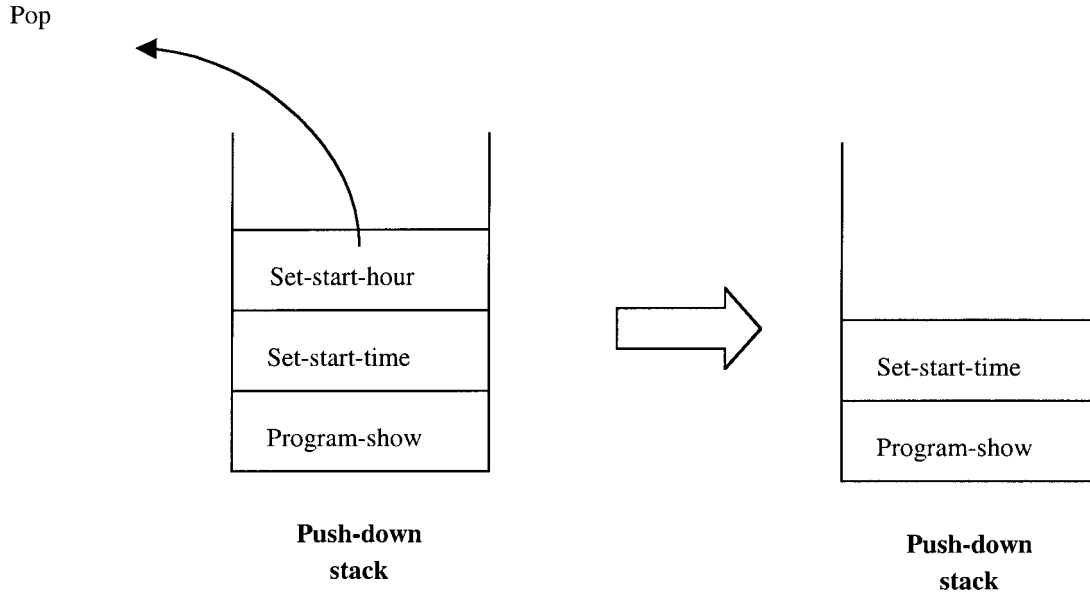

Figure 6. The completed element SET-START-HOUR is popped off the pushdown stack.

number of actions in each trial. The results indicate that the grammars were successful in capturing most of the variations of the action sequences. Although the high percentage of fit may be an artifact of the task, it does show that simple grammar rules, which correspond roughly to the elementary edit operations, are sufficient to capture the similarities between the many possible variations of the action sequences.

Action protocols from a blocks world task. To test the generality of ACT-PRO, we evaluated the grouping program with another set of action protocols collected from 48 subjects (16 subjects per condition) as they worked on a blocks world task. In the task, subjects had to copy eight colored blocks shown in the target window (T) to the workspace window (W), using the colored blocks in the resource window (R). All the windows were covered by black windows. To do the task, the subjects had to uncover the windows one at a time. The three conditions differed in the perceptual–motor cost of uncovering that target window (see Fu & Gray, 2000b). What we were interested in was whether ACT-PRO could capture the sequence of window uncoverings throughout the task. Table 4 shows the sequences of window uncoverings in which we were interested. Table 5 shows an example of the grammars used to capture the sequences.

Each subject had to finish 40 trials, for a total number of 1,920 trials in the data set. The total number of window uncoverings was 55,296. Thirteen grammars were constructed, each of them representing a strategy that we believed the subjects used. The grouping program calculated the percentage of window uncoverings captured by the grammars in each trial. Out of 1,920 trials, the worst-fitting match was 68.7%, the best-fitting match was 90.6%, and the overall average match was 82.5%. All the variations of the strategies represented by the 13 grammars (strategies) were captured, suggesting that the grammar representation was successful at capturing the variations in the window-uncovering sequences for the task.

Table 4
The Basic Window-Uncovering Sequences (Strategies) for the Blocks World Task That Are Represented as Grammars in the Grouping Program

Window-Uncovering Sequences
TRTW
TRTRTW
TRTWTW
TRWRWRW(RW)
TRTRWRWRW(RW)
TRWRWRW
TRTRWRWRW
TRWRW
TRTRWRW
TRW
TRTRW
(TW)
RTW

Note—T, target window; R, resource window; W, workspace window. Recurrent patterns are in parentheses. Therefore (TW) denotes any number of repetitions of the pattern TW—for example, TWTW, TWTWTW, and so forth.

Table 5
A Sample Grammar Used for the Blocks World Task

Grammar TRTW:
TRTW: [Object1][Object2][Object3][Object4]
[Object1] → *Target*
[Object1] → *Target* [Object1]
[Object2] → *Resource*
[Object2] → *Resource* [Object2]
[Object3] → *Target*
[Object3] → *Target* [Object3]
[Object4] → *Workspace*
[Object4] → *Workspace* [Object4]

By inspecting the window-uncovering sequences that were not captured by the grammars, we found that all involved strategies that we did not expect. For example, all 13 grammars were built assuming that the subjects had to look at the patterns of color blocks in the *target* window before they could start copying. However, some subjects started copying color blocks without looking at the *target* window, presumably doing so by pure guessing. These strategies were therefore not captured by any of the 13 grammars we constructed.

The action sequences required to change a setting on the perverse VCR were much more constrained by the interface than were the sequence of window uncoverings in the blocks world task. The perverse-VCR interface limited the number of possible combinations of actions that would successfully change a setting. For the blocks world task, the interface itself does not impose any constraints on the window-uncovering sequence—each of the windows can be uncovered independently of the others. The patterns of window-uncovering sequences emerged purely out of the task characteristics and the processing limits of the subjects.

The results of ACT-PRO on two different data sets show that the syntactic representation used for grouping is powerful enough to capture most of the variations of action sequences. The performance of the grouping program depends on how structured subject's behaviors are and how homogenous these structures are across different subjects. These factors, in turn, depend on how many constraints are imposed by the task, the interface, and the processing limits of the cognitive system on subjects' behaviors.

The Tracing Program

The tracing program partitions and maps the action stream to the elements in the hierarchical structure and provides the measures that facilitate validation of the researchers' theory. The set of action protocols collected as subjects programmed the simple VCR interface was used (Gray & Fu, 2000). The hierarchical goal structure in Figure 1 is constructed on the basis of one constructed from previous research (Gray, 2000). Action protocols were collected from 72 subjects; each of them programmed four shows. Similar to the perverse-VCR interface, the subjects had to program each show correctly twice in a row before they could move on to the next show. This cre-

ated 1,044 correct trials, with a total of 36,877 actions. These actions were parsed into 12,761 goals and subgoals by the tracing program of ACT-PRO. The validation program identified 148 (1%) push mismatches and 810 (6%) pop mismatches. The goodness of fit was therefore 93%. The high level of fit to the postulated goal hierarchy supports the hierarchy's psychological validity. Indeed, in Gray and Fu (2000), the researchers were not interested in the behaviors that matched the goal hierarchy so much as they were in interpreting mismatches to the goal hierarchy.

USES OF ACT-PRO

The Grouping Program

ACT-PRO will not induce action patterns, but it can be used to determine how much of the data can be accounted for by patterns specified by the researcher. The two data sets we used represented different levels of analysis. The actions patterns captured in the perverse-VCR interface (buttonpresses) had a finer grained level than those in the blocks world task (window uncoverings). In the perverse-VCR interface, we were interested in explaining each action made. On the other hand, in the blocks world task, we were interested in the strategies used by the subjects. The different grains of analysis were dealt with by building grammars that matched the grain size of the actions that are passed to the grouping program. The results showed that the grouping program could handle different levels of analysis equally well.

Another use of the grouping program is to identify changes in how the behavior streams are partitioned as subjects, with practice, move along the skill dimension with practice from problem solving to skilled behavior. As their skill increases, the size of recursive action patterns grows until it reaches the processing limits of the subjects (Card et al., 1983). One way to keep track of this change is to build grammars that capture recursive action patterns of different sizes. For early trials, the best-fitting grammars should be those that capture fewer numbers of actions. As the subject progresses, the best-fitting grammars will be those that capture more actions. By looking at how the best-matching set of grammars change across trials, the effect of practice can be studied.

The Tracing Program

ACT-PRO will not induce hierarchical structure, but it can be used to compare the fit of alternative hierarchical structures or, as in the example provided above, to obtain a goodness-of-fit measure on a single hierarchical structure. Hierarchical structures are useful in characterizing subjects' behaviors in different tasks or interfaces. For example, if the researchers are interested in how different interfaces might induce different strategies (and hence different hierarchical behavioral patterns), alternative hierarchical structures can be built and matched to the subjects' behaviors. Although hierarchical goal structures were used in the examples provided above, ACT-PRO is not limited to validation of goal hierarchy. In fact, ACT-PRO can handle any hierarchical structures constructed by the researchers, as long as the hierarchical structures can be represented by a set of pushdown stacks.

Care must be taken in evaluating the measure of goodness of fit. Violations of a hierarchical structure are not necessarily evidence that the structure is not good. Violations might represent transfer among alternative hierarchical structures, systematic errors owing to cognitive limits interacting with the design of the task, and so on (e.g., see Fu & Gray, 2000a; Gray, 2000; Gray & Fu, 2000).

Finally, an intended application of ACT-PRO is in the development and validation of analytic models. In building such models, ACT-PRO can be used to interrogate existing data to ensure that the structures built into the model (groups as well as hierarchies) are constrained to those found in the data. In evaluating the model, ACT-PRO can be applied to action protocols collected from the model, and the results can be compared with action protocols collected from human subjects. For complex tasks, this use of ACT-PRO may provide a means of comparing a model with data that is more detailed and finer grained than is typically possible.

LIMITATIONS

The main limitation of ACT-PRO is that it is only capable of top-down analyses. Researchers have to specify the patterns and/or hierarchy in advance. Another limitation is that since ACT-PRO works on the assumption that structures (or action patterns) exist in subjects' behavior, the usefulness of ACT-PRO depends on whether the interaction of the subject and the task does indeed induce these structures, as well as in how homogenous such structures are across different subjects.

FUTURE EXTENSIONS

ACT-PRO is best suited to confirmatory analyses that require the researcher to specify in advance the patterns that they seek. One extension of ACT-PRO would be to incorporate exploratory sequential data analysis techniques to facilitate the generation of grammars. Another potential extension would be to make use of the latency information in the action protocols (i.e., the interaction interval) in both the matching and the parsing process. This use of latency would be particularly useful when the researcher is interested in validating a precise cognitive model that predicts not only the sequence of actions, but also latencies between actions.

CONCLUSIONS

Tools for matching action protocol data to theory are rare. A large volume of fine-grained action protocols is easy to collect, but the analysis of the data is time consuming and tedious. For the three data sets described in the evaluation section, the building of grammars, on av-

erage, took the researchers 2–3 h, and the average running time was about 1 h. If the data sets were analyzed one by one manually, it would have taken weeks or months of analysis time. Using the metric of analysis time (AT) to sequence time (ST) ratio (Sanderson & Fisher, 1994), the AT:ST ratio for using ACT-PRO is about 1:10 (the ST for each of the data sets was about 30–40 h). The AT:ST ratio for hand-coding the trials reported in Gray (2000) is estimated to be 100:1. The reduction in AT by using ACT-PRO is therefore quite significant.

We have shown that ACT-PRO is useful in helping researchers analyze discrete action protocol data. ACT-PRO automates the process of grouping and matching the hierarchical structure to the vast amount of action protocols collected from multiple subjects. We believe that the process that ACT-PRO automates is central to a broad range of research interests. We believe that, with minor, task-specific modifications, ACT-PRO can be applicable to many other tasks.

REFERENCES

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering* (pp. 1-8). Los Alamitos, CA: IEEE Computer Society Press.

Bakeman, R., & Gottman, J. M. (1997). *Observing interaction: An introduction to sequential analysis* (2nd ed.). New York: Cambridge University Press.

Bakeman, R., & Quera, V. (1992). SDIS: A sequential data interchange standard. *Behavior Research Methods, Instruments, & Computers*, **24**, 554-559.

Bakeman, R., & Quera, V. (1995). *Analyzing interaction: Sequential analysis with SDIS and GSEQ*. New York: Cambridge University Press.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human–computer interaction*. Hillsdale, NJ: Erlbaum.

Fu, W.-T., & Gray, W. D. (2000a). *Implications of rational analysis for interface design: Observations in a natural learning environment*. Manuscript submitted for publication.

Fu, W.-T., & Gray, W. D. (2000b). Memory versus perceptual–motor tradeoffs in a blocks world task. In *Proceedings of the Twenty-Second Annual Conference of the Cognitive Science Society* (pp. 154-159). Hillsdale, NJ: Erlbaum.

Gray, W. D. (2000). The nature and processing of errors in interactive behavior. *Cognitive Science*, **24**, 205-248.

Gray, W. D., & Fu, W.-T. (2000). The influence of source and cost of information access on correct and errorful interactive behavior. In *Proceedings of the Twenty-Second Annual Conference of the Cognitive Science Society* (pp. 663-668). Hillsdale: NJ: Erlbaum.

Magnusson, M. S. (2000). Discovering hidden time patterns in behavior: T-patterns and their detection. *Behavior Research Methods, Instruments, & Computers*, **32**, 93-110.

Olson, G. M., Herbsleb, J. D., & Rueter, H. (1994). Characterizing the sequential structure of interactive behaviors through statistical and grammatical techniques. *Human–Computer Interaction*, **9**, 427-472.

Pitkow, J. E., & Pirolli, P. (1999). Mining longest repeated subsequences to predict World Wide Web surfing. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems* (pp. 120-124). Berkeley: University of California Press.

Sanderson, P. M., & Fisher, C. (1994). Exploratory sequential data analysis: Foundations. *Human–Computer Interaction*, **9**, 251-317.

Sankoff, D., & Kruskal, J. B. (Eds.) (1983). *Time warps, string edits, and macromolecules: The theory and practice of sequence comparison*. Reading, MA: Addison-Wesley.

Simon, H. A. (1975). The functional equivalence of problem solving skills. *Cognitive Psychology*, **7**, 268-288.

Simon, H. A. (1996). *The sciences of the artificial* (3rd ed.). Cambridge, MA: MIT Press.

NOTES

1. The complete descriptions of the goal hierarchy, the task, and subject behaviors are in Gray (2000) and Gray & Fu (2000). The goal hierarchy shown in Figure 1 is simplified.

2. Although the example here shows a hierarchical structure of goals, ACT-PRO is not limited to this. ACT-PRO is neutral and can handle different kinds of hierarchical structures.

3. The outputs of the grouping and tracing programs are linked to a spreadsheet program (Excel 98) to facilitate display of the results.

(Manuscript received November 16, 2000;
accepted for publication February 6, 2001.)