Behavior Research Methods & Instrumentation
1974, Vol. 6, No. 2, 167-170

SESSION VII
SYMPOSIUM: THE SKED SYSTEM FOR PROGRAMMING AND RECORDING BEHAVIORAL EXPERIMENTS

ARTHUR G. SNAPPER, *Western Michigan University*, **Presider**

State notation and SKED: A general system for control and recording of behavioral experiments*

RONALD M. KADDEN
FDR Veterans Administration Hospital, Montrose, New York 10548

State notation is a language for describing behavioral procedures and data acquisition formats. A minicomputer system has been developed for translating state notation into operating computer programs, which can control 10 simultaneous and independent experiments. A description of the system is provided, including the hardware necessary to interface the computer with the experimental environment

Programming minicomputers in machine language can be quite time-consuming, and must be done with great care when it involves on-line process control and data acquisition. This is especially true when several independent processes are being controlled simultaneously by a single computer. At many installations, heavy demands are placed on a minicomputer by research scientists and graduate students, with no experienced computer programmer available to produce the complex programs that arise from ongoing research projects. Even if such services were generally available, the demands for on-line computer time are usually so great that only bits and pieces of free time scattered throughout the day and evening are available for programming. To simplify the problem of program complexity and to make it possible to work within available time, a minicomputer system (SKED) was developed for programming experimental procedures. Although the system described here has been implemented only on the PDP-8 series (Digital Equipment Corporation) of computers, the same principles would apply to any other computer.

The system is based upon the mathematical theory of sequential switching circuitry, using the state notation model. This notational language was chosen for three primary reasons: (1) It is precise enough for designing electronic switching circuitry, (2) it is simple enough to be easily learned, and (3) it is already being used by engineers and scientists to describe a wide variety of sequential devices and procedures. The notation is independent of the nature of the sequential events considered and is also independent of particular hardware or software systems. With it, a basic scientific requirement is satisfied: accurate communication, both among investigators working in different areas within a field, and also across disciplinary lines. As for its usefulness within the field of experimental psychology, the notational language has been shown to completely and unambiguously describe behavioral procedures (Snapper, Knapp, & Kushner, 1970; Snapper, 1973). The language of state notation consists of a set of precisely defined elements called states, operators on the elements called transitions, and outputs. Through the appropriate interconnection of the state elements, behavioral procedures involving discrete stimuli and responses can be notated.

Figure 1 is a flow diagram of the steps required to complete an experiment, from formulation of the experimental problem through analysis of the resultant data. Before a final diagram of the experimental procedure can be drawn in the notational language, the experimental question must be precisely defined. It has been our experience that notating the procedure as it is developed by the E facilitates completion of the final version. Since the notation requires that all procedural details be stated explicitly, ambiguities in the experimental design become apparent quickly and therefore tend to be resolved early. The notational language also facilitates communication among co-Es, eliminating debates about procedural details that may have been poorly communicated from one scientist to another.

*This work was supported by the National Institute of Mental Health, under Grant MH-13049 to W. N. Schoenfeld, and by the Veterans Administration Hospital, Montrose, New York.

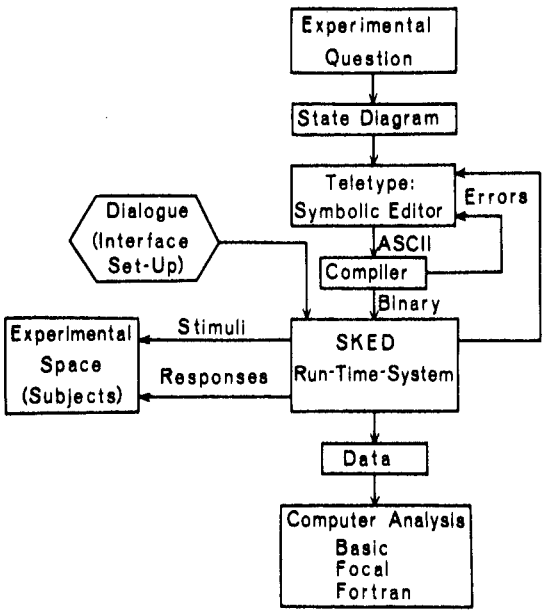

Fig. 1. Flow chart of procedures required to complete an experiment.

After a final version of the experimental procedure has been notated as a state diagram, a three-stage process is necessary to implement it on the computer (vide, Snapper & Kadden, 1973; Snapper & Walker, 1971). In the first stage, the diagram is preserved on some storage medium (paper tape, magnetic tape, magnetic disk, etc.) by means of an editing program. In the second stage, the stored diagram is processed by a compiler program which produces a machine language state table (a binary representation of the original state diagram), which is also preserved on the storage medium. A third stage involves committing the computer to on-line control by means of an executive program or run-time system (RTS) that reads the machine language version of the program and enters it for one or more stations.

The first stage involves a straightforward transformation from a free-hand drawing of the state diagram to a linear Teletype format. The state diagram is entered into a symbolic editor program, which provides considerable power for adding to, modifying, and rearranging the text as entered on the Teletype. The symbolic editor is also used to correct errors detected in the next stage by the compiler, and additionally to modify the original state table to provide parametric variations of the basic procedure. The editor does not detect errors in the state diagrams, or even "typos," but only stores text as provided by the teletypist. Once a complete, accurate version of the state diagram has been provided to the editor, it is preserved on the desired storage medium in the American Standard Code for Information Interchange (ASCII). It is this stored version that will be entered into the compiler in the next stage, and will also serve as the basis for reediting at any future time.

The compiler, in the second stage of the programming process, translates the state diagrams stored by the editor into state tables, in a binary format compatible with the (RTS). Since the compiler reads the state diagrams directly from the storage medium, the only information required of the E at this stage is the number of data recording counters the completed program will require. The compiler is programmed to detect certain illegal characters and formats and undefined variables. These errors are listed on the Teletype as they are detected, and correction of them requires return to the symbolic editor program. The output of the compiler is preserved on the selected storage medium, ready for entry into the RTS.

The SKED RTS is the executive program which monitors the progress of individual state tables and which distributes response inputs and stimulus outputs between interface connections and the appropriate state tables. Each cycle of the RTS program is triggered by a pulse from a free-running external clock. The RTS causes the portions of any state tables that are time-dependent to be advanced by one unit step. In this way, several independent state tables, representing separate experimental procedures, are serviced by each clock pulse. Similarly, at each clock input, all response input lines are examined to determine whether responses occurred in any of the operative experimental stations since the last clock pulse. These are directed by the RTS toward the appropriate state table for determination of possible consequences (e.g., increasing the response ratio count, reinforcement delivery, data recording, etc.). Any stimulus outputs resulting from the interaction of response and clock inputs with the state tables are directed by the RTS toward the appropriate interface output connectors. The RTS also provides other services to the state tables, such as data recording, advancing of states within state sets, and intercommunication between state sets within a single state table. In addition, provision has been made and a format provided for programming special-purpose routines for the RTS, such as to generate a random number sequence on-line, to perform arithmetic or higher-order mathematical functions, to provide unique data recording formats, or to service specialized input-output devices. Both these and the standard routines are stored by the RTS and are made available to each state table as required.

The RTS program format remains unchanged for all state tables, although the program does have two modifiable features. The first of these is the special-purpose routines, referred to above, which may be added as needed at the end of the RTS. The second modifiable feature of the RTS is the input-output section. Since various laboratories have different interface configurations, and since within the same laboratory the same input-output interface channels may be required to perform different functions, at different times, the SKED RTS has been written with a flexible input-output section. The configuration of this section is determined by a program called the dialogue, which

assigns the available input and output channels to experimental stations. Thus, in a laboratory with two 12-bit input words and three 12-bit output words, approximately 10 experimental stations can be accommodated, each with two input (response) lines and three or four output (stimulus) lines. More stations could be accommodated by assigning fewer lines per station, or, alternatively, all 24 response and 36 stimulus lines could be assigned to a single station. With more input or output words, even greater flexibility would be possible. The input-output section of the RTS could be modified by the dialogue each time a new state table is written, or the configuration could be left unchanged through many experiments. For this reason, a different shape of enclosure was placed around the dialogue element in Fig. 1, to indicate that it is available for use whenever desired, but need not necessarily be used each time a new experiment is begun.

Data recording requirements are specified as part of the state diagram at the time the diagram is typed, using the symbolic editor. Each event whose occurrence is to be recorded, clock pulses or responses, causes a specified data recording counter in the computer memory to be incremented once for each occurrence of the event. Special recording requirements, such as interresponse time distributions, response latency distributions, responses in subintervals of fixed time cycles, etc., can be accommodated as part of the state diagram. Since data recording is treated by the notation like a stimulus output, as a pulse incrementing a location in the computer's memory, data recording routines can be expressed in state notation just as the reinforcement contingencies are. Thus, data recording requirements are expressed as an integral part of the notation used to specify the reinforcement schedule.

The progress of an experiment can be monitored at any time during a session by manually typing a request for a data dump. Alternatively, frequent automatic data dumps could be specified in the initial state diagram so that all the data recording counters for a particular experiment would be listed on the Teletype or on some other storage medium at prespecified intervals. If the purpose of these data dumps is to monitor the progress of an experiment, the data recording counters may be left unmodified after each dump, so that each successive dump will represent cumulative progress up to that time. However, the use of frequent data dumps during a session can accomplish another purpose as well. If an E wished to collect large amounts of data, such as successive interresponse times or successive cardiac interbeat intervals, for example, the data recording area in memory would be so large as to leave no space for the running of any other experiments in the computer at the same time. This problem can be solved by reserving a fixed-length data recording area, dumping the data frequently, resetting all counters in the data area to zero after the dump, and starting to record data from the beginning of the data area after each dump. Thus, successive data dumps can provide a continuous listing of the desired data throughout an experimental session. When operated in this mode, the RTS stores the data to be dumped in a temporary buffer storage area and immediately clears all counters in the primary data collection area, so that new data can be collected without interruption, even while previous data are still being dumped. This is particularly useful where data storage must be done on paper tape, involving relatively long dump times.

Presently, data analysis, such as tests of correlation, statistics, etc., must be performed off-line, using standard data treatment packages like BASIC, FOCAL, or FORTRAN. These cannot be used on-line as background low-priority programs with the SKED RTS as a foreground high-priority program, since the data analysis programs occupy considerable memory space that would probably be needed for state tables and data recording counters. Also, their operations would be made extremely slow by the high rate of interrupts generated by the free-running clock. For these reasons, data are preserved on a storage medium by the RTS, and are read off-line at a later time for analysis.

Some of the problems due to limitations of the size of available memory can be alleviated by adding more memory fields. Such additions allow the use of more state tables (i.e., more experiments running simultaneously), longer state tables, and expanded data recording capabilities. The SKED RTS comes in two sizes, one for memories consisting of only 4K words, and a second that allows the addition of 4K memory fields up to a maximum of 32K words. The number of available memory fields is specified in the dialogue program when other hardware configurations are also specified. When state tables are subsequently loaded into memory, they are placed by the RTS into the first space available in memory, the uppermost memory fields being utilized only after the lowest ones have been filled. If, at the time a new state table is about to be loaded, it is determined that one or more of the preceding ones in memory have been terminated as active state tables, all remaining active state tables are first moved down to fill the vacant space, and only then is the new table entered, above its predecessors. It might also be noted here that if the same experiment is to be run simultaneously, but independently, on four Ss, only a single state table need be edited and compiled. At the time of loading the state table, the four stations in which it is to be run are specified, and the RTS will load the program four times in successive portions of memory. In this way, four procedurally identical, but wholly independent, experiments can be run simultaneously.

Figure 2 shows a block diagram of the hardware required for a typical system. Indicated at the top of the figure are the computer and its affiliated devices for storing and entering programs and state tables, and for receiving and preserving data dumps. All communication with the experimental Ss is accomplished through the

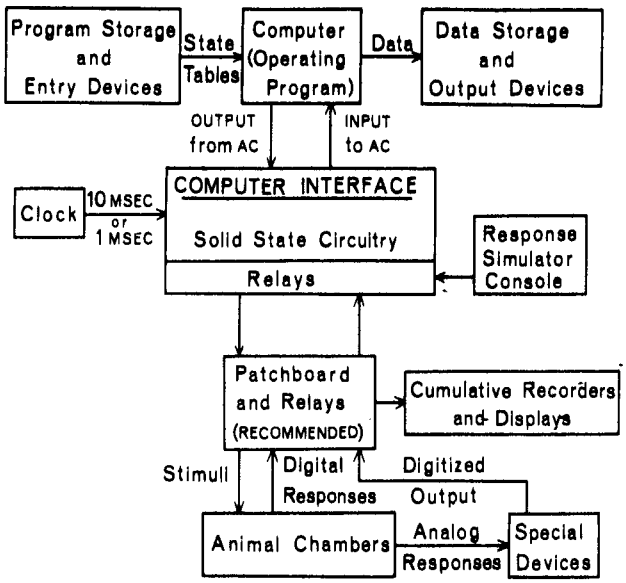

Fig. 2. Block diagram of equipment configuration.

accumulator (AC) of the computer, which transmits pulses to operate or terminate stimuli and receives signals indicating the occurrence of responses. All signals which communicate with the experimental Ss also pass through an interface composed of solid state signal conditioning circuitry and general-purpose relays. A free-running clock is connected to the solid state portion of the interface. The repetition rate of this clock is typically set for 10-msec interpulse intervals (100 Hz), although, where a higher rate is required for finer temporal analysis, 1-msec intervals (1,000 Hz) can be used, if only one S is run at a time. The higher clock rate is also useful to speed up the process of debugging new state tables. Another device, which is useful for debugging both programming and circuitry problems, is a response simulator panel with a pushbutton which can operate each input line. Although responses can also be entered on the Teletype, much higher rates are possible with pushbuttons. Indicator lights are also useful on input and output lines.

Another adjunct to the system hardware which is very helpful, although not required, is a patchboard interface with general-purpose relays. Routing signals through such a setup provides flexibility for changing the configuration of experimental stations and for adding new equipment as acquired. It can save many hours of changing connections, and its systematic format helps to minimize errors when rewiring becomes necessary. The relays associated with this interface further add to its flexibility and also act to protect the computer interface from overloading. All stimuli, responses, and connections to peripheral data recording devices are channeled through the patchboard interface. They are connected directly to the computer interface when no patchboard interface is available.

The SKED RTS requires digital responses for its operation. Although some computer systems have analog-to-digital conversion capabilities, their operation requires considerable sacrifice of computer time. It is therefore preferable to perform conversions outside of the computer, and to send only digital signals to the RTS. These signals might include electrocardiograms, blood pressure, electroencephalograms, or the output of force-sensitive levers. The outputs of each A-to-D conversion device would be transmitted on as many channels as necessary as separate inputs to the RTS.

A system such as the one described here can be started small, being added to in piecemeal fashion as funds become available. With even a minimal operational system, no more than 5 man-hours should be required to produce a state table, including drawing the state diagram, editing, compiling, specifying the interface configuration by means of the dialogue program, debugging, and reediting and recompiling to correct errors. This time can be reduced for simple experiments or where a library of basic state tables and interface configurations is available, but may be extended where considerable rearrangement of the system hardware is required.

REFERENCES

Snapper, A. G. Use of a notation system for digital control and recording. Behavior Research Methods & Instrumentation, 1973, 5, 128-131.

Snapper, A. G., & Kadden, R. M. Time-sharing in a small computer based on a behavioral notation system. In B. Weiss (Ed.), *Digital computers in the behavioral laboratory*. New York: Appleton-Century-Crofts, 1973. Pp. 41-97.

Snapper, A. G., Knapp, J. Z., & Kushner, H. K. Mathematical description of schedules of reinforcement. In W. N. Schoenfeld (Ed.), *The theory of reinforcement schedules*. New York: Appleton-Century-Crofts, 1970. Pp. 247-275.

Snapper, A. G., & Walker, A. The SKED software system. Digital Equipment Computer Users Society Program Library, 1971, No. 8-465.