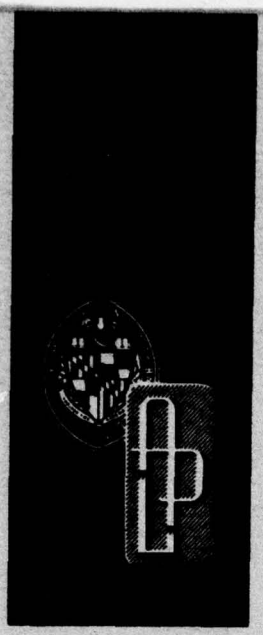
AD-A057 395 JOHNS HOPKINS UNIV LAUREL MD APPLIED PHYSICS LAB F/G 14/2
THE ART OF SIMULATION: SOME TOOLS AND TECHNIQUES DEVELOPED AT T--ETC(U).
MAR 78 L R GIESZL N00017-72-C-4401
UNCLASSIFIED APL/JHU/TG-1316 NL

1 OF 1
ADA
057395

END
DATE
FILMED
9-78
DDC

APL/JHU
TG 1316
MARCH 1978
Copy No. 4



(12)

LEVEL II

Technical Memorandum

THE ART OF SIMULATION: SOME TOOLS AND TECHNIQUES DEVELOPED AT THE APPLIED PHYSICS LABORATORY

by L. R. GIESZL

THE JOHNS HOPKINS UNIVERSITY ■ APPLIED PHYSICS LABORATORY

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

PLEASE FOLD BACK IF NOT NEEDED FOR BIBLIOGRAPHIC PURPOSES

REPORT DOCUMENTATION PAGE

1. REPORT NUMBER: TG 1316

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (*and Subtitle*): (16) The Art of Simulation: Some Tools and Techniques Developed at the Applied Physics Laboratory

5. TYPE OF REPORT & PERIOD COVERED: Technical report

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s): (10) L. R. Gieszl

8. CONTRACT OR GRANT NUMBER(s): (15) N00017-72-C-4401

9. PERFORMING ORGANIZATION NAME & ADDRESS:
The Johns Hopkins University Applied Physics Laboratory
Johns Hopkins Rd.
Laurel, MD 20810

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Z02A

11. CONTROLLING OFFICE NAME & ADDRESS:
Naval Plant Representative Office
Johns Hopkins Rd.
Laurel, MD 20810

12. REPORT DATE: (11) Mar 78

13. NUMBER OF PAGES: (12) 40 p. 43

14. MONITORING AGENCY NAME & ADDRESS:
Naval Plant Representative Office
Johns Hopkins Rd.
Laurel, MD 20810
(9) Technical memo,

15. SECURITY CLASS. (*of this report*): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: NA

16. DISTRIBUTION STATEMENT (*of this Report*)

Approved for public release; distribution unlimited.

(14) APL/JHU/TG-1316

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

Simulation
Model
ASTN
Automata
Network
Decision tables
Hierarchical interrogation
Event-store

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

In this paper two devices are used to narrow the communications gap between the end users of large-scale computer reliant models and the analysts who must find, review, or create such models. The first device involves the use of examples to describe the significant characteristics of three simulation techniques: event-store, ASTN, and hierarchical interrogation. The second device makes use of multiple equivalences. In this case, the conversion from one modeling technique to another (which shows the two to be equivalent) is prompted by a consideration of some salient features within the process to be simulated. Some of the original work on these techniques has been done at APL. The subliminal nature of these techniques and the causal implicative chains leading to their implementation is discussed from a user-oriented point of view. A section on decision tables and a decision table processor ~~developed at APL~~ is included to assure a clear treatment of hierarchical decision table interrogation.

DD FORM 1 JAN 73 1473

031 650

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

APL/JHU
TG 1316
MARCH 1978

Technical Memorandum

THE ART OF SIMULATION: SOME TOOLS AND TECHNIQUES DEVELOPED AT THE APPLIED PHYSICS LABORATORY

by L. R. GIESZL

THE JOHNS HOPKINS UNIVERSITY ■ APPLIED PHYSICS LABORATORY
Johns Hopkins Road, Laurel, Maryland 20810
Operating under Contract N00017-72-C-4401 with the Department of the Navy

THE JOHNS HOPKINS UNIVERSITY
APPLIED PHYSICS LABORATORY
LAUREL, MARYLAND

ABSTRACT

In this paper two devices are used to narrow the communications gap between the end users of large-scale computer reliant models and the analysts who must find, review, or create such models. The first involves the use of examples to describe the significant characteristics of three simulation techniques: event-store, ASTN, and hierarchical interrogation. The second device makes use of multiple equivalences. In this case, the conversion from one modeling technique to another (which shows the two to be equivalent) is prompted by a consideration of some salient features within the process to be simulated. Some of the original work on these techniques has been done at APL. The subliminal nature of these techniques and the causal implicative chains leading to their implementation is discussed from a user-oriented point of view. A section on decision tables and a decision table processor developed at APL is included to assure a clear treatment of hierarchical decision table interrogation.

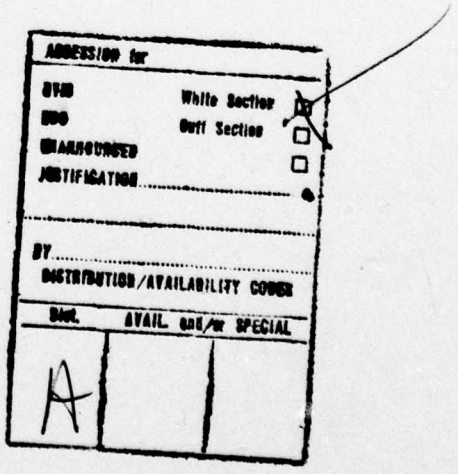


TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1. INTRODUCTION

The term "simulation" is used in such a generic sense with a multiplicity of specifically intended meanings that a communications gap is beginning to widen between potential users and implementing analysts. The problem has created such confusion among the analysts themselves that a number of tutorial papers have been written on the differentiating aspects of various forms of simulation techniques (see Refs. 1 through 5). Since the majority of these papers appear to be addressed to the analyst, there has been little guidance for the naive user. The purpose of this report is to provide, in some measure, such guidance.

CATEGORIES AND TECHNIQUES

Since simulation in general is such a broad subject, partitioning it into categories and then breaking down each category into techniques, facilitates understanding. One such breakdown is as follows:

1. Discrete Simulation, a model that changes in discrete steps
    a. Coded analytical model
    b. Finite automata

Ref. 1. H. Pople and G. Werner, "An information Processing Approach to Theory Formation in Biomedical Research," Proc. Spring Joint Comput. Conf., AFIPS Press, 1972.

Ref. 2. J. G. Laski, "On Time Structure in (Monte Carlo) Simulations," Oper. Res. Q., Vol. 16, No. 3, 1965.

Ref. 3. P. J. Kiviat, "Development of Discrete Digital Simulation Languages," Simulation, February 1967.

Ref. 4. A. Pritsker and N. Hurst, "GASP IV: A Combined Continuous-Discrete Fortran-based Simulation Language," Simulation, September 1973.

Ref. 5. T. Ören, "Software for Simulation of Combined Continuous and Discrete Systems: A State-of-the-Art Review," Simulation, February 1977.

c. Event centered model

d. Activity centered model

e. Process oriented model

2. Continuous Simulation, a model with continuous dependent variables

a. Differential equation model

b. Analog model

c. Approximate with discrete model

3. Combined Continuous/Discrete

a. Straightforward coded procedure

b. Discrete method augmented by continuous facilities

c. Discrete continuous portions

d. Combined-system simulation language

Since the intent of this report is to provide an insight into the art of simulation, a systematic discussion will not be attempted for all of the above techniques. In fact, several papers have been published (Refs. 3, 4, and 5) to provide depth, even at the expense of initial insight. This report will cursively present characterizations of some of the above techniques; then, with the aid of specific definitions, it will present a discussion narrowed down to the combined continuous/discrete category.

CORRELATIONS AND EQUIVALENCES

Conceptually, it would seem that the first two categories listed above should be proper subsets of the third, i.e., the combined category. This is an assumption implicit in listing techniques involving an interface between the discrete and continuous notions. In practice, the pure continuous models are rarely seen. An analyst may call his model discrete even though it has several processes containing continuous variables; a war game with continuous dynamic motion but discrete detection events is such an example. The blurring of definitions makes the restriction to "combined continuous/discrete" academic or semantic.

Three distinct techniques used in combined continuous/discrete simulation are so dissimilar in approach that they appear to be incompatible. They are:

1. Event-store (event centered);

2. Augmented state transition network (ASTN) (finite automata); and

3. Hierarchical decision table interrogation.

They will be shown, in an empirical fashion, to be virtually equivalent. This equivalence results in the narrowing of the communications gap between simulation users and designers. A user may picture his problem as a network while an analyst might use an event-store model to solve the problem. Technique equivalence, and especially the process involved to show equivalence, gives a freer choice in the "problem-to-model" step in writing a simulation.

One technique uses decision tables and relies on an efficient method of implementing them on a digital computer. For this reason the uses and requirements of decision tables are discussed in this report. No other documentation is available on the large decision table preprocessor created several years ago at the Laboratory to help "modularize" the process of writing a computer-reliant war game.

The process of showing several simulation techniques to be equivalent should result in a better understanding of each technique. Similarly, the method of using a decision tree analogy of a decision table to show a form of processing optimality may give a better understanding of decision tables and how they can be used.

2. CONTEXTUAL BACKGROUND

LENGTH OF THE CAUSAL CHAIN

Modeling and simulation provide possible solutions when a problem is characterized by long chains of logical, causal conditions or a large number of component interactions. Modeling eventually becomes a necessity as the causal chain or number of interactions reaches a critical level. The actual critical level may be determined by any of the following: the existence or lack of a "pattern" to the problem, the availability of easy data processing implementation, the limited capacity of the human mind to connect all the variables, the time available to work on the problem, or (even) the amount of sophistication one wants to associate with the implementation of a solution. A simple observation derived from experience, but self-evident on reflection, is that the more necessary one finds a model, the more difficult and time consuming becomes the task of representing the problem with a model. Also, as a model becomes more necessary, a computer-reliant model, or simulation, tends to be the solution.

MODEL VERSUS SIMULATION

The distinction between model and simulation is given by Pople and Werner (Ref. 1) as a distinction between a "hypothesis model" (the big picture in the mind of the investigator) and a "simulation model" (the computer program that represents the investigator's attempt to codify the big picture). A successful simulation is usually the result of an investigator hierarchically iterating through these two "models." Reference 1 reports an attempt to make the iterative process more automated. The procedure is based on beginning with an implicative or causal net as the input to a computer program to help in the detailed specifications of a hypothesis model (since it is generated by a computer it can be immediately incorporated into a simulation model). In contrast, the prime concern in this report is the intuitive creation of the representing causal net. The process involved here is what eventually determines whether a simulation is really necessary and the type of simulation required.

The literature on simulation contains many references to "discrete simulation," "continuous simulation," and even "combined continuous and discrete simulation." Each of these categories has origins in specific types of problems. A continuous simulation is used to

model the various transport equations (differential equations) of some continuous process; a discrete simulation is used to model a process that is described with a schedule of events occurring in some discrete fashion. Most simulations fall into a combined category since a part of most computer models is best described by continuous equations, but must use discrete time occurrences of events. The event-store technique devised by APL's Planning and Analysis Group (PAG) falls into this category.

3. EVENT-STORE SIMULATION

For more than 15 years PAG has made use of a simulation technique called "event-store" programming. The technique has been a vital component of many computer-reliant war games; an understanding of the method is critical to understanding how such games are played (see Ref. 6). Although this programming technique has become more sophisticated as computers have become more complex, it has not been formally documented as an independent computing package. Traditionally, an understanding of the method has been a side benefit of using a particular program. Here an attempt is made to fill the documentation gap. An example, given in the next section, shows the relationship of this modeling to network and rule-oriented modeling.

Although the event-store programming used by PAG came prior to most simulation languages such as GPSS, SIMSCRIPT, and CSL, it has many of the features of these languages. In fact, if one is familiar with GPSS, he would already have a familiarity with the underlying concepts involved in PAG event-storing. Although this method predated most formal work on queues and stacks in the current jargon of computer science, it is now easily explained in these terms.

DEFINITIONS

To avoid semantic difficulties, definitions of various terms used for describing a computer-reliant simulation are presented below:

System - The "thing" (process or engagement) to be modeled;

Entity - An identifiable object of interest used in the description of the system;

Attribute - The property or value of an entity;

Activity - A process that causes changes in the system;

State of the System - A list of all entities, attributes, and activities that gives a complete description of the system at a given point in time;

Ref. 6. A. Andrus and R. Meier, "Naval Air Strike Model, PAM 18A," APL/JHU report, June 1962.

Continuous Systems - Systems whose changes are described by continuous equations;

Discrete Systems - Systems whose changes are described by a set of discrete values;

Event - A request for a specific activity;

Time Step - A specified time interval between allowable changes in the state of the system.

MOTIVATION FOR REPRESENTATION

The first problem encountered in simulating an activity with actual computer code is that of defining an algorithm that captures the essence of the activity. The algorithm may be in the form of causal net, a logical set of rules, or a collection of equations. Since there is never a single or unique method of algorithmic representation for a process, part of the simulation art is to determine the simplest representation. The next problem is implementation. A simple English language algorithm may be very difficult to reproduce directly into computer code. The definition, and redefinition, of the problem should direct the choice of complexity needed for a model and the most appropriate type of simulation technique.

FIRST EXAMPLE

A simple example will be used to clarify the point. Suppose we want to simulate an old-fashioned gun duel. After the duelists are in place, each antagonist fires at his opponent as quickly as possible after a prearranged signal. Each then determines if his opponent is hurt. If no one has been hit, they fire again. Actual firing times can be determined by adding a small random time increment to each man's receipt of the signal. The observational delay can be simulated as a small random variation. With these assumptions, we can diagram (or flow chart) the process, see Fig. 1.

THE PARALLEL PROBLEM

The flow chart, Fig. 1, seems very simple. It is, as a process description, but the method of implementing it as an algorithm suitable for a (serial) computer is not immediately obvious. The

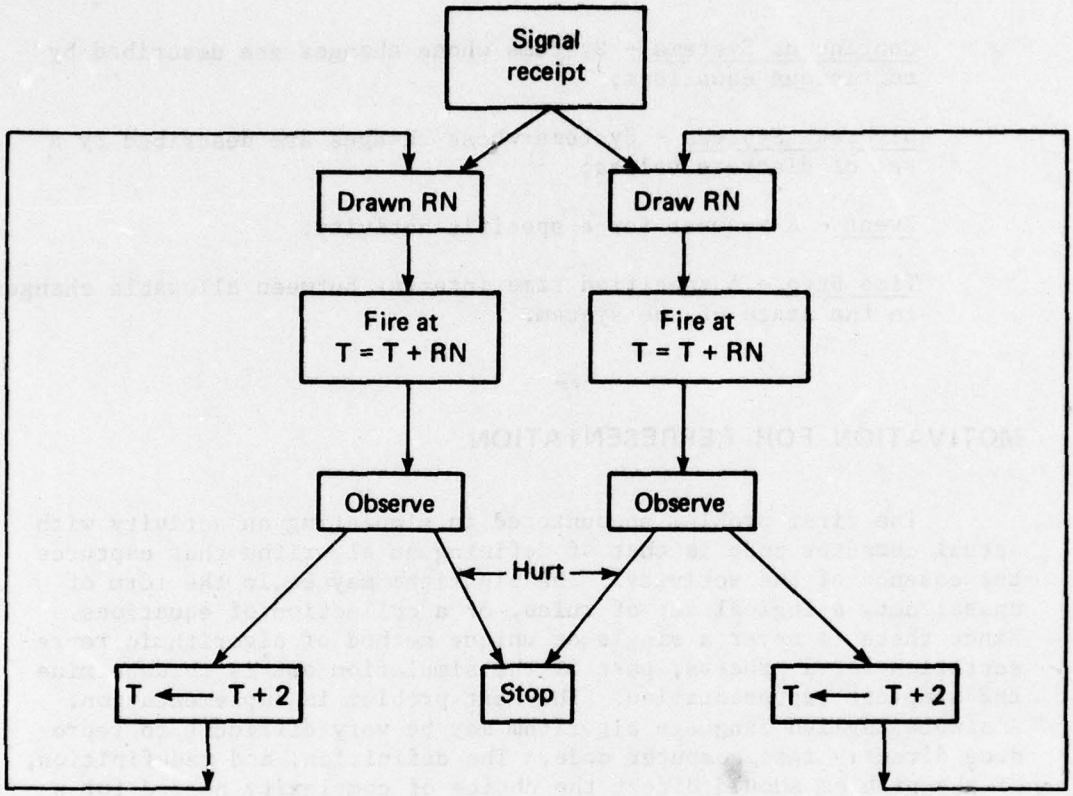


Fig. 1 Example of the Simulation of an Activity

problem is two-fold: first, there are <u>two</u> processes operating in parallel (both men fire); second, actions are scheduled for a time in the future. The second problem is seen in "fire at T = T + RN" (that is, prepare to fire at a new time that is current time plus a random number of seconds), and in "T ← T + 2" (i.e., set the "clock" for the one who fires up by 2 s). The first problem implies that some logical conversion (or detailed specification) of the flow chart must be made to allow the emulation of a parallel process on a computer that only performs sequential chains of computations. It involves specifying all possible actions along with the associated actors and times and considering the various interactions.

ITS SOLUTION

The simplicity of the example allows a direct implementation by paying careful attention to the time-ordering implied by the

relative magnitudes of the random numbers drawn and the association maintained between T_1 and actor one (and T_2 and actor two). Figure 2 is a more computationally oriented diagram of the model (it is assumed that the FIRE subroutine includes a decision capability that allows for the probability of a hit and passes the hit/not hit? information to control). Much of the computation is devoted to "remembering" who is firing. It should be noted that the possibility of a refire by one man preceding the initial firing of the other is precluded in the restriction of RN ($0 \leq RN \leq 1$) along with the "normal" 2-s observation time. If these restrictions were lifted, i.e., if one man could fire and refire before the other, this same model would handle the proper ordering of fire (eliminate the 2-s addition to T_1 or T_2).

EVENTS

So far, "events" and "event storing" have not been mentioned. The simplicity of the model represented in Fig. 2 makes it unnecessary to formalize events or their queues to transform the representation into computer code. It should be evident that if an event must be defined for this process (from an intuitive consideration of the original problem, or by following the actual flow of control through Fig. 2) it would be FIRE. The FIRE routine is called at various time intervals that are unknown until the appropriate random numbers are drawn. This characteristic, which could have gone unnoticed in a simple coding from the next flow chart, Fig. 3, is the original item that instigated the quantum jump from ordinary (iterative) programming to event-store programming.

A simple English language algorithm of the dueling problem from the point of view offered by this "event-store frame of mind" might be as follows:

1. Draw RN1, RN2, and set T_1 = RN1, T_2 = RN2;
2. Put requests for FIRE in a queue, Q, tagged for J = 1, T = T1; and for J = 2, T = T2 in time (T) order;
3. Take earliest FIRE request from Q, and execute according to its subject tag (J); set T = tagged time;
4. If a HURT condition arises, go to step 8; otherwise proceed;
5. Draw a new RN, set $T_i = T + 2 + RN$ (for i = J);
6. Store a new FIRE request in Q (tagged with i and T_i);

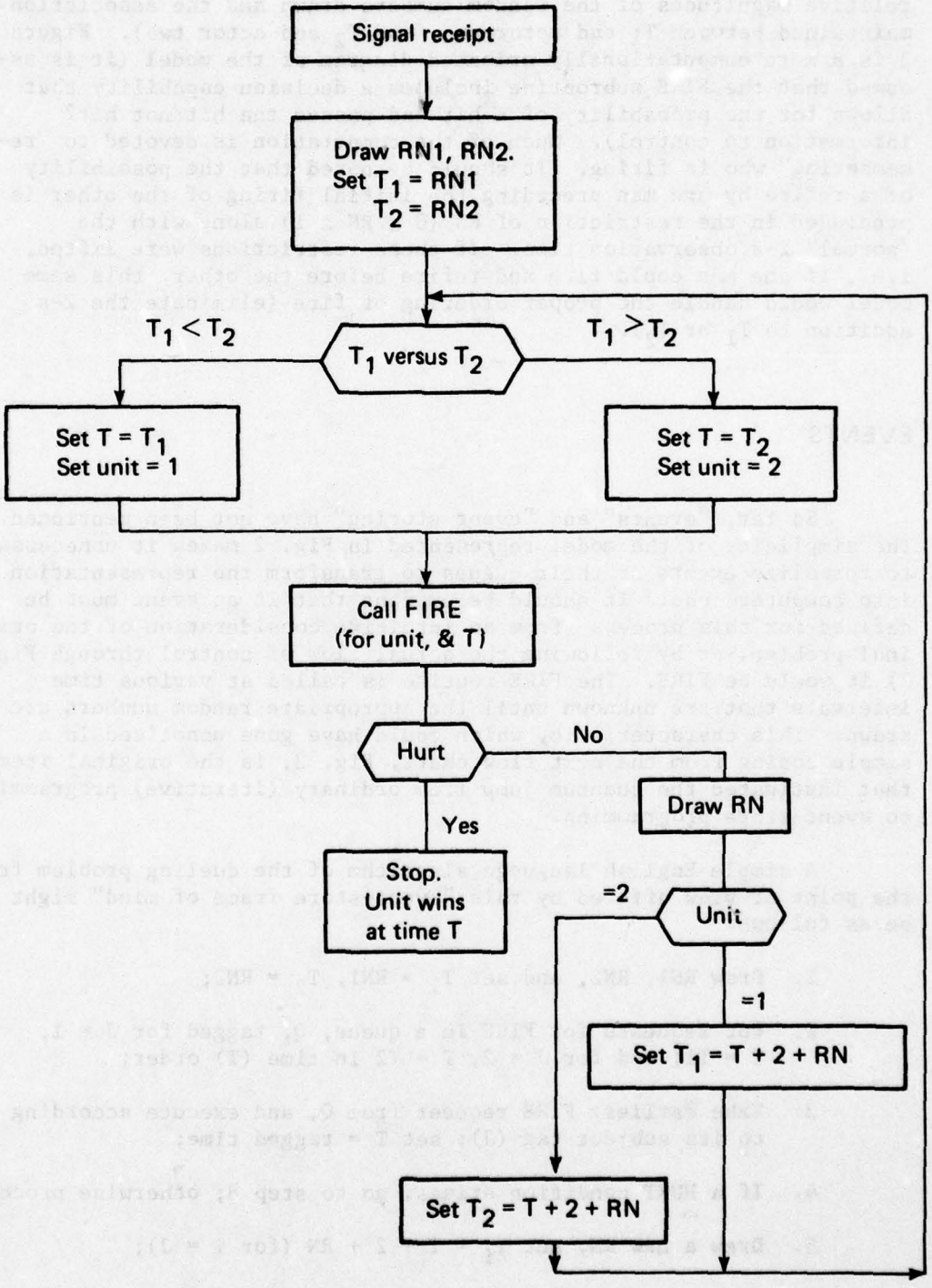


Fig. 2 Computer Flow Chart of Simple Duel

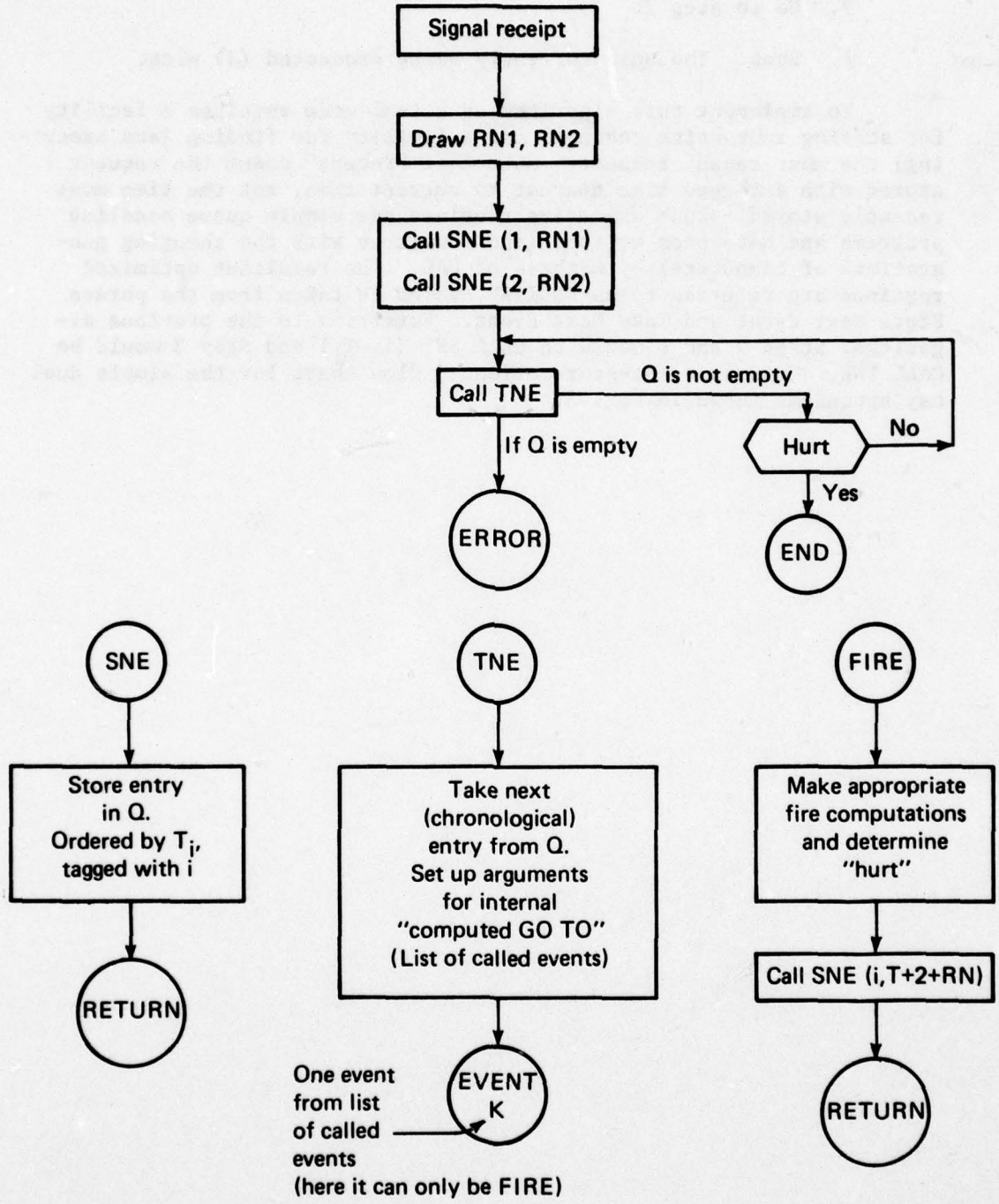


Fig. 3 Simple Duel (Event-Store) Representation

7. Go to step 3;

8. Stop. The unit currently being processed (J) wins.

To implement this algorithm in actual code requires a facility for storing subroutine requests and a facility for finding (and executing) the most recent request. Note that "recent" means the request stored with a tagged time nearest to current time, not the time most recently stored. Such executive routines are simple queue handling programs and have been written (and rewritten with the changing generations of computers) by members of PAG. The resultant optimized routines are referred to as SNE/TNE, which is taken from the phrase Store Next Event and Take Next Event. Referring to the previous algorithm, Steps 2 and 6 would be CALL SNE (i, T_i) and Step 3 would be CALL TNE. Thus an event-store oriented flow chart for the simple duel may appear as shown in Fig. 3.

4. ASTN, A SECOND TECHNIQUE

AUGMENTED STATE TRANSITION NETWORK

The process used to illustrate the event-store concept was somewhat contrived to maintain simplicity. However, the addition of more interactions or restrictions to the modeled problem may make this event-store concept the most convenient technique for implementing an actual computer program. To show the essential characteristic evolving from the complexity of a problem, a second example will be analyzed. The second example will also be used to illustrate the equivalence of event-store, network simulation, and hierarchical decision table interrogation. The approach to an event-store model formulation is basically a descriptive, forward, chronological ordering of a process protocol. The other two forms of simulation techniques considered here are characterized by a backward, result-to-cause analysis.

DESIGNING A SIMULATION

The usual procedure for outlining a simulation amenable to immediate computer implementation is to formalize a fixed scenario, decide on the events involved in the depicted process, and then let the logical event causal chains determine the proper sequencing of events. This was done previously in a simple example, and will be repeated for one that is more complex to illustrate the "essence" of event-store programming. For comparative purposes, the network implementation will be derived from this event-store characterization (vice the process description). Similarly, the decision table technique will be derived from the network formulation. We do not imply that it is necessary to proceed through a sequence of specifically documented steps of analysis to reach one of the last two implementation techniques. Since some of the thought processes involved in proceeding from a model to an event-store simulation sometimes appear to be almost on a subliminal level (you see it clearly one moment, then you're not so sure), it seems possible that this sequence for program development may indeed be the natural thought process, whether the steps are documented or even consciously recognized. The slight intrusion of psychology into a paper on computer programming is caused by concern that traditionally the "event-store concept" has been elusive to many end-product users; and the ASTN or Decision Table Interrogation forms of simulation may be more subject to misunderstanding because of their result-to-cause characteristic. Such a characteristic will be clearly noted in the following example.

SECOND EXAMPLE

A "duel-'til-death" process will be modeled in the proper format for each technique. It differs from the previous duel because the effects of various types of projectile damage are considered in greater detail. If the duelists keep shooting until one of them dies (vice a single wound before) then the following four events can occur for each:

1. Instant death,
2. Death within 1 minute,
3. Death within minutes, or
4. Inhibited from further participation.

In the fourth event, either duelist either runs out of ammunition or loses his capability to fire. Note that this event is not independent of the others. It will be seen that the techniques considered do not require independent events.

The scenario for this process will be as follows:

1. Each participant has an inherent shooting skill, PH(i), which equals the probability of i hitting his opponent;
2. Probabilities, P(J), are given to be associated with the above events; i.e., given that participant i was hit, then P(J) equals the probability that event J will occur for i;
3. Each participant fires, observes, then fires again. The initial firing is a random increment after T = 0. Refiring is a random variation, about 10 s from the previous shot (of the same participant).

With such a simple process description, the temptation to show the difficulties involved in a straightforward coding, as was done for the first problem in Fig. 2, is hard to suppress. However, this can be dismissed as obfuscation disguised as illustration. A few moments of thought on the increased amount of bookkeeping (keeping track of events in the order that they occurred) which these four events would require for a flow chart, such as Fig. 2, should convince the reader that the likelihood of easy event-store implementation is needed. Hence the solution to the problem obtained by the three simulation techniques are given without further justification.

EVENT SPECIFICATION AND CAUSATION

The scenario suggests that three types of events will characterize the process. There should be a FIRE event, a DIE event, and an INH (become inhibited) event. These are the only activities in which either opponent may engage. Each duelist schedules himself for a FIRE event; but he is scheduled, if the odds fall that way, for the other events by his opponent. Since there are two opponents, the actor for (or subject of) an event is uniquely specified by noting who stored the event. If i stored a FIRE event, then i fires when that event comes up. If i stored a DIE or INH event and that event comes up, then it is i's opponent who either dies or becomes inhibited. Figure 4 is a diagram showing an event-store approach for this problem. Take Next Event (TNE), an executive routine previously explained, is characterized here as a three-way switch. It is also assumed that there is an associated time for each of the originally specified events (e.g.,

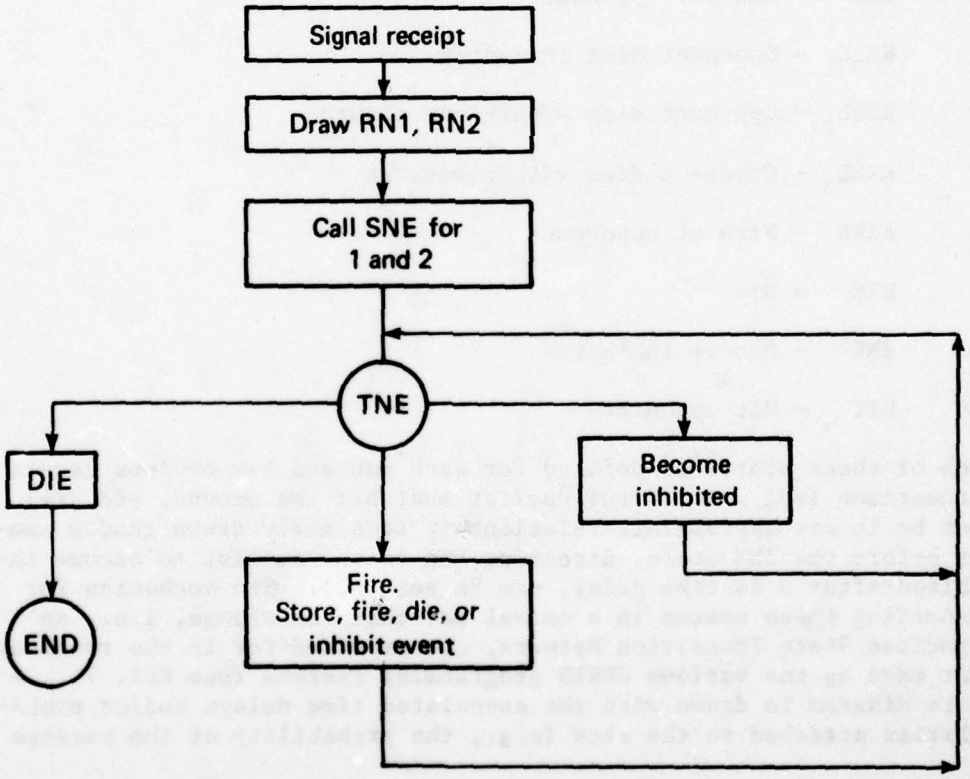


Fig. 4 Event-Store Characterization of "Duel 'til Death"

"instant death" would be a DIE event stored to come up at current time, "die within one minute" would a DIE event stored to come up by T + 60 seconds and so on).

RESULT-TO-CAUSE CHARACTERIZATION

To proceed from this formulation to a network formulation it is necessary to reconsider the features of the model from the viewpoint of result-to-cause. There are various "things" that can occur. These things are states in a special form of automaton, or network. Each man can fire, each man can die, and each man can cause an "injury." An injury can be characterized as a thing, or state, in accord with the original four process events. Thus, to set up a state diagram, the appropriate states are:

INH - Inhibit opponent

$KILL_1$ - Opponent dies instantly

$KILL_2$ - Opponent dies within one minute

$KILL_3$ - Opponent dies within minutes

FIRE - Fire at opponent

DIE - Die

INH' - Become inhibited

HIT - Hit opponent

Each of these states is defined for each man and has obvious causal connections (e.g., the first duelist must hit the second, and P(4) must be in the appropriate relationship to a newly drawn random number before the INH state, directing the second duelist to become inhibited after a $\Delta 4$ time delay, can be entered). The mechanics for connecting these states in a causal net that can change, i.e., an Augmented State Transition Network, are provided for in the routines that make up the various GERTS programming systems (see Ref. 7). A state diagram is drawn with the associated time delays and/or probabilities attached to the arcs (e.g., the probability of the passage

---

Ref. 7. A. Pritsker and W. Happ, "GERT: Graphical Evaluation and Review Technique: Part I - Development," J. Ind. Eng., Vol. 17, No. 5, May 1966.

from state 1 to state 2 is designated on the arc connecting node 1 to node 2). A source node is the one where processing originates; sink nodes are nodes (or states) where all diagram processing stops. Figure 5 is an example of network implementation of the duel-'til-death process. The arc labeling on the figure should not cause confusion, although some labels refer to delay times and others to probabilities. A dotted line between two nodes indicates the possibility that a node (the one not present in the mainstream of the network) will replace another. For example, node n_{12} or node n_{13} may replace node n_{11}. An arrow, tipped with a node number, but no node, is used in this illustration to imply a node change order. If node n_{31} is activated, the arrow from it to the node number n_{23} (labeled $\Delta 4$) implies that node n_{23} will be brought into the active network, replacing node n_{21} after delay time $\Delta 4$.

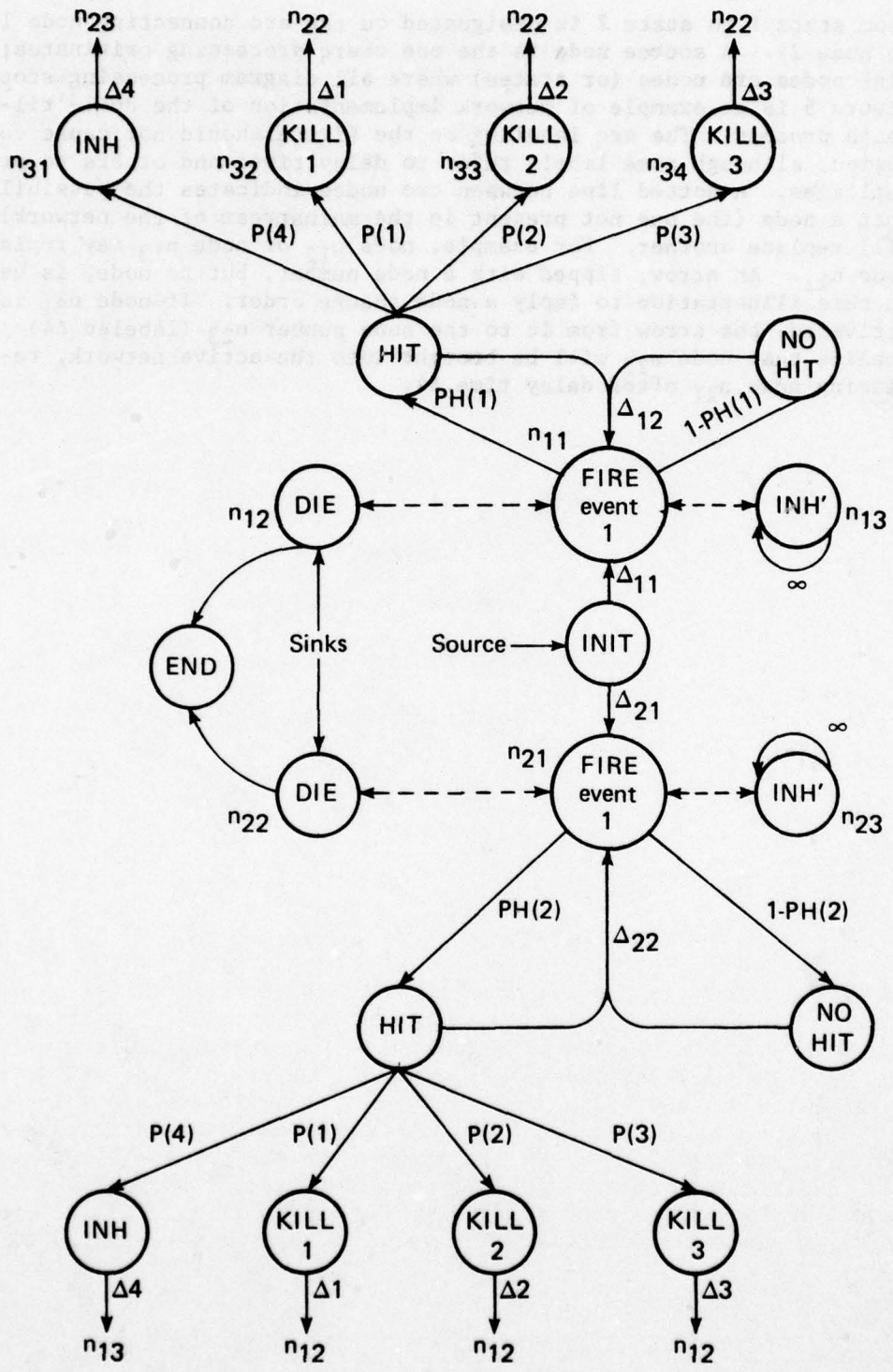

n23
Δ4
INH
n31
n22
Δ1
KILL 1
n32
n22
Δ2
KILL 2
n33
n22
Δ3
KILL 3
n34
P(4)
P(1)
P(2)
P(3)
HIT
NO HIT
PH(1)
Δ12
1-PH(1)
n11
FIRE event 1
n12
DIE
INH'
n13
∞
Δ11
END
Sinks
Source
INIT
Δ21
∞
n21
FIRE event 1
DIE
n22
INH'
n23
PH(2)
Δ22
1-PH(2)
HIT
NO HIT
P(4)
P(1)
P(2)
P(3)
INH
KILL 1
KILL 2
KILL 3
Δ4
Δ1
Δ2
Δ3
n13
n12
n12
n12

Fig. 5 An ASTM

5. DECISION TABLES AND AN APL/JHU DECISION TABLE PROCESSOR

The third simulation technique that will be shown (by an example) to be equivalent to the event-store and ASTN involves iterative interrogation of properly defined decision tables. The example used for this illustration of equivalence is the duel-'til-death problem mentioned above. In fact, the network characterization will be converted to a decision table interrogation simulation. In order to avoid semantic pitfalls in this process, it seems appropriate to provide the proper background of definitions and techniques associated with the use of decision tables.

DECISIONS, TREES, OR TABLES

Let us suppose that a person decides on a particular action, A, because of specific facts, say F_1, F_2, F_3, and F_4. These "facts" are actually a current evaluation of corresponding conditions (C_1, C_2, C_3, and C_4). For example:

"A if F_1 and F_2 and not F_3 and F_4"; or

"A if C_1 is yes and C_2 is 28 and C_3 is no and C_4 is 5".

Similar expressions in Boolean terms as causal strings are:

$$C_1 \wedge (C_2 = 28) \wedge (\sim C_3) \wedge (C_4 = 5) \rightarrow A,$$

or, $$C_1 \wedge (C_2 = 15) \wedge (\sim C_3) \wedge (C_4 = 10) \rightarrow A,$$

or, $$(\sim C_1) \wedge (C_2 = 0) \wedge (C_3) \rightarrow A$$

The problem with such alternative causal strings is noticed when communicating such a decision rule to someone:

"Action A occurs if:

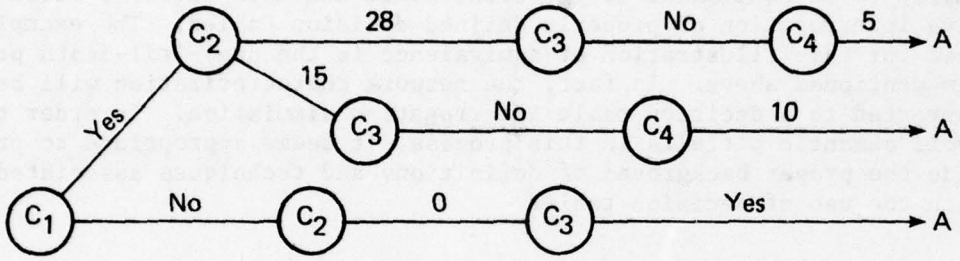
C_1 is yes, C_2 is 28, C_3 is no, and C_4 is 5;

or, C_1 is yes, but C_2 is 15 (and C_4 is 10), and C_3 is still no;

or, C_1 is no, C_2 is 0, and C_3 is yes".

One solution for documenting such a decision rule in a simple manner is to put it in the form of a decision tree.

A decision tree has conditions for nodes, their evaluation for branches, and actions for leaves. The example given above could be drawn as:



The notation assumes that if a path exists from the C_1 node to the leaf A, then action A is specified. If no such path exists (i.e., $C_2 = 7$), then action A is not taken.

A decision table is another method of documenting a decision that involves alternative combinations of condition evaluations. It is simply a tabulation of all possible paths in the corresponding decision tree. The most practical form of decision table (for computer applications) is the limited entry table. This just adds the requirement that the conditions written in the tabular form require only yes/no answers (i.e., C_2 above becomes $C_2 \stackrel{?}{=} 28$, $C_2 \stackrel{?}{=} 15$, and $C_2 \stackrel{?}{=} 0$). The more specific conditions are written in the top rows of the decision table; the appropriate actions are written in the bottom rows. Various decision alternatives, or rules, are specified as Y, N, - (yes, no, and don't care) in the columns on the right side of the table. Each column spells out the values of the specific conditions required to determine all actions denoted by Y in the corresponding action row.

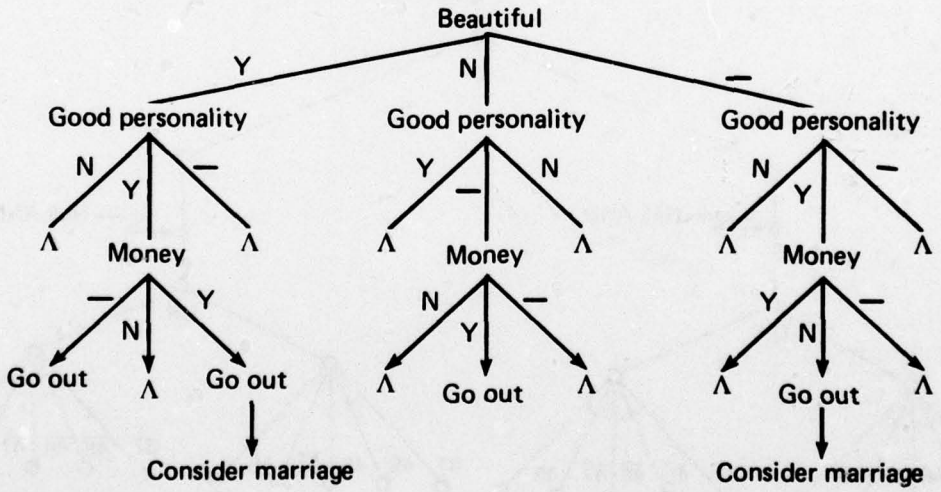
Figure 6 is another simple (chauvinistic) example of a decision table and a decision tree corresponding to a decision rule involving confusing alternatives. A decision is made using the tree by following the path corresponding to the value at each encountered node. A decision is made using the decision table by evaluating the conditions and then consecutively interrogating each rule (reading the column and matching it against the condition evaluation column). If no rule is satisfied, then the table is exited and no further action is taken.

If a girl is beautiful and has a good personality, I will go out with her. If she also has money, I will consider marrying her. However, if she is ugly, but has money, I will go out with her, but won't consider marrying her. If she has a good personality and she is poor, I will go out with her, and may even marry her.

(a) In English

		Rules			
Actions	Beautiful	Y	Y	N	–
	Good personality	Y	Y	–	Y
	Has money	–	Y	Y	N
Conditions	Go out	Y	Y	Y	Y
	Consider marriage		Y	N	Y

(b) Decision Table



(c) Decision Tree

Fig. 6 Three Forms for a Decision Process

PROCESSING EFFICIENCY

If a list of decision tables must be interrogated a large number of times, or if a large number of decision tables must be interrogated, it must be assumed that a minimal amount of time is spent on each nonsatisfied decision table. This points out the necessity of a "good" decision table processor for converting the original tables to very efficient executable subroutines. Such processors exist; one was developed at the Laboratory for the purpose. It generates a Fortran subroutine that has been optimized for fast negative exits. For example, all conditions that are required (either yes or no) constant through all rules are evaluated and compared, before even evaluating other conditions required for the table. If any one condition fails, the subroutine is immediately exited. There are, of course, the other usual masking techniques that are found in most decision table processors.

The specifics of the decision table processor are given under the heading "A Decision Table Processor," but it seems appropriate to show, by example, the development of efficiency. It should be self-evident that a decision table can be represented as an AND/OR tree. Figure 7 shows such a representation of a decision table G0 (for the

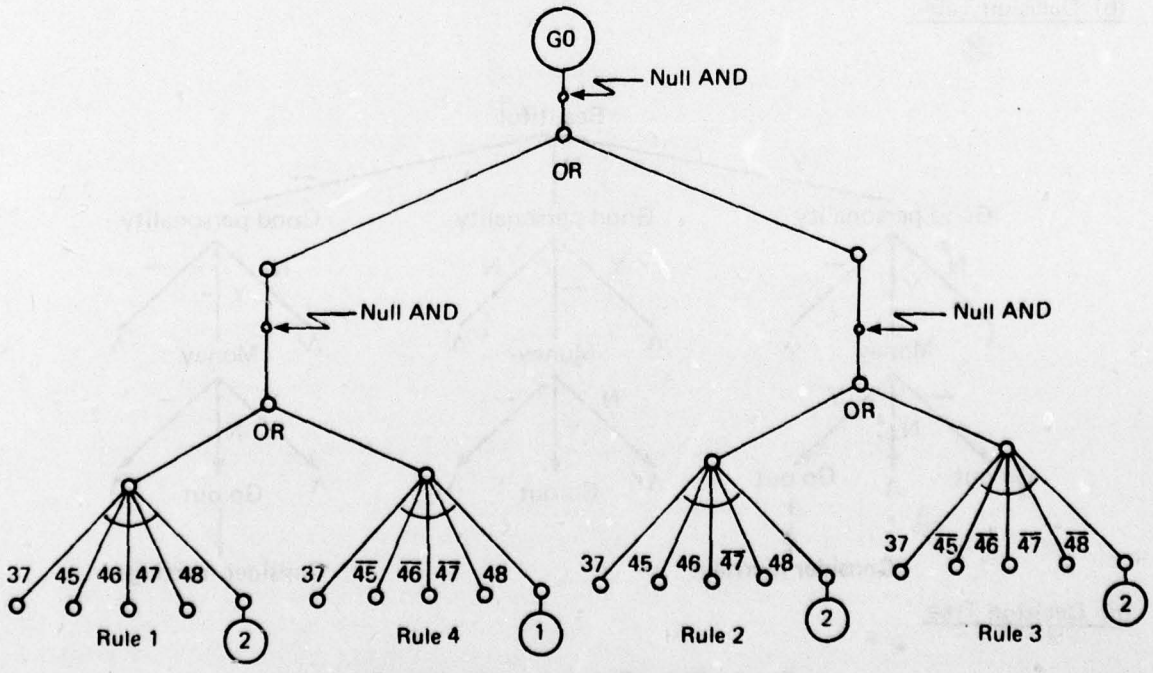


Fig. 7 Tree Representation of Decision Table G0

medical diagnosis of epilepsy), with four rules and five conditions. A traversal of the tree implies either first- or second-stage epilepsy. Equivalently, in order to rule out epilepsy, we must assure the impossibility of traversing the tree shown in Fig. 7, i.e., the whole tree must be scanned for a possible traversal.

Tree searching can be done efficiently if the number of branches that must be searched are minimized. The specific "fast exit" efficiency of a processed decision table can be assured if the table is derived from a minimal decision tree. Since minimizing the branches of a tree is easier to visualize, our examples will refer to decision trees, but the results will be associated with equivalent decision tables. It is obvious that by shifting conditions (i.e., branches) to higher level "AND-nodes" whenever possible, we can ensure early rejection of major passageways through the tree. For example, in Fig. 8, condition 37 (since it was common to all rules) was placed in an AND relationship with the remainder of the four rules. If condition 37 is not satisfied, the tree cannot be traversed because of the initial AND relationship. Hence consideration of the tree (decision

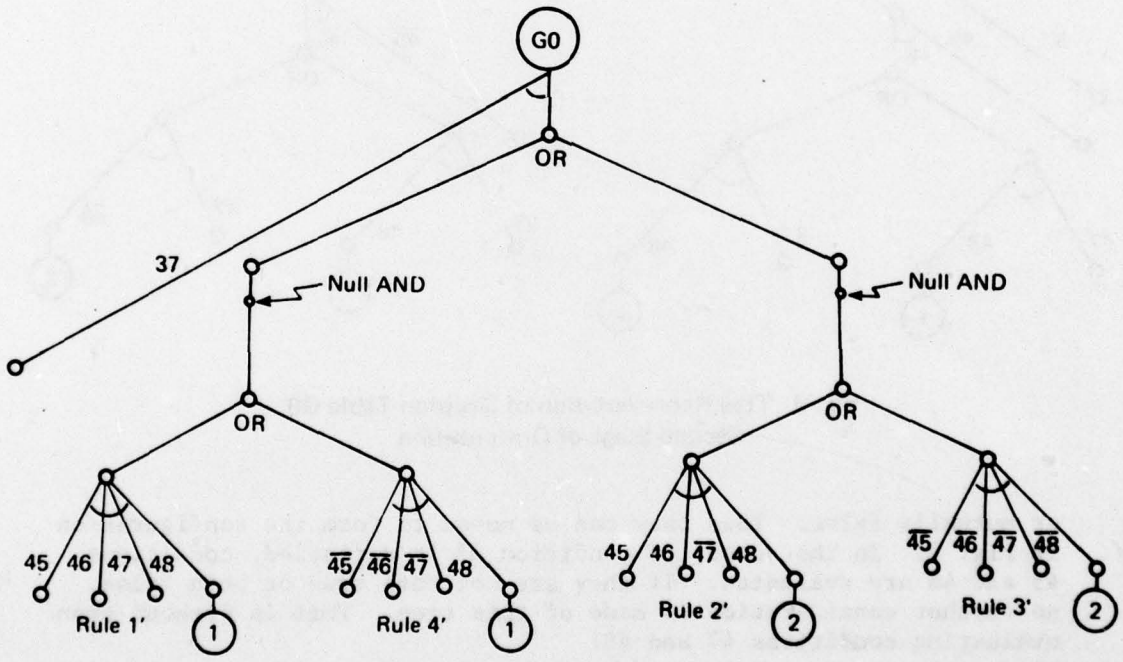


Fig. 8 Tree Representation of Decision Table G0, First Step of Optimization

table) may involve no more than the evaluation of a single condition before control is passed on to the next tree.

Similarly, if a group of conditions occurs in all rules, and each rule is either all true or all false, then they may be combined and moved to a higher AND node. In the above example it can be noted that, in all four rules, conditions 45 and 46 appear as mutually true

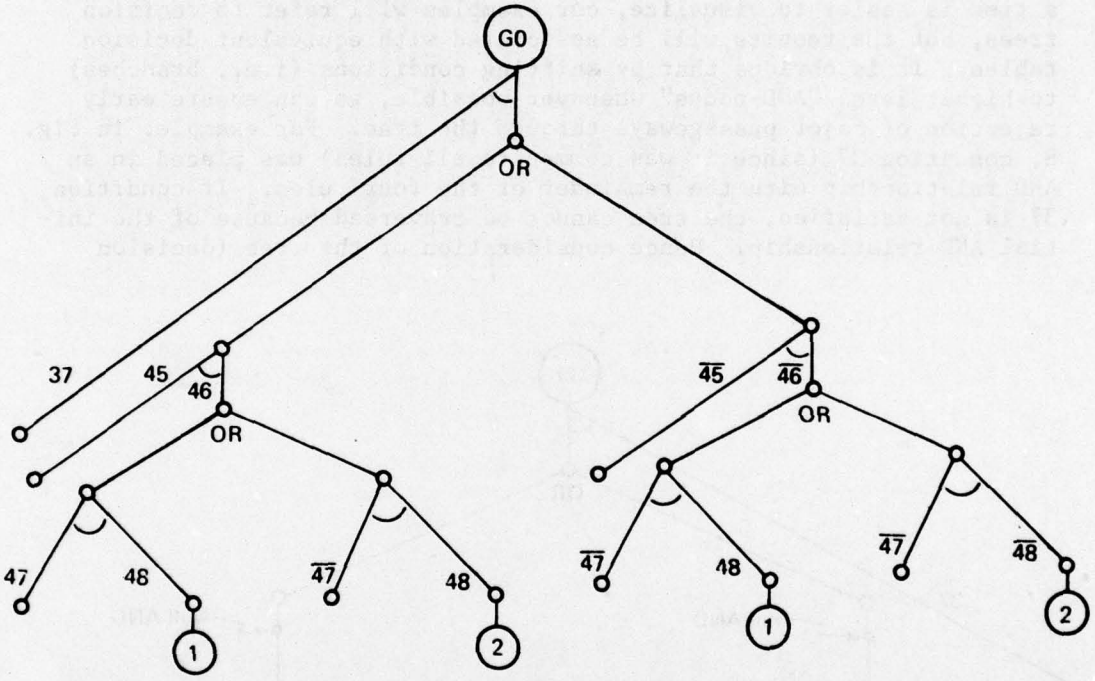


Fig. 9 Tree Representation of Decision Table G0, Second Stage of Optimization

or mutually false. Thus they can be moved to form the configuration in Fig. 9. In that case, if condition 37 is satisfied, conditions 45 and 46 are evaluated. If they are not both true or both false, no further consideration is made of this tree. This is without even evaluating conditions 47 and 48!

Since both of these optimizing features are incorporated in the decision table processor, it does not seem unwarranted to call the processed routines efficient. The unprocessed and processed G0 decision tables are given in Figs. 10 and 11, respectively.

```
 1 *DCTR G0  5 4 2                                                    G0     1
 2       SUBROUTINE G0                                                G0     2
 3 C   DECISION TABLE FOR EPILEPSY                                    G0     3
 4 C                                                                  G0     4
 5       COMMON/NEUR1/CND(100),TUMOR(10),ISW(20)                      G0     5
 6 *COND 1                                                            G0     6
 7       IF (CND(37).EQ.1) CALL T$SY                                  G0     7
 8 *COND 2                                                            G0     8
 9       IF (CND(45).EQ.1) CALL T$SY                                  G0     9
10 *COND 3                                                            G0    10
11       IF (CND(46).EQ.1) CALL T$SY                                  G0    11
12 *COND 4                                                            G0    12
13       IF (CND(47).EQ.1) CALL T$SY                                  G0    13
14 *COND 5                                                            G0    14
15       IF (CND(48).EQ.1) CALL T$SY                                  G0    15
16 *RULE 1   Y Y Y Y N   Y N                                          G0    16
17 *RULE 2   Y Y Y N Y   N Y                                          G0    17
18 *RULE 3   Y N N N N   N Y                                          G0    18
19 *RULE 4   Y N N N Y   Y N                                          G0    19
20 *ACTN 1                                                            G0    20
21       WRITE(6,8001)                                                G0    21
22 *ACTN 2                                                            G0    22
23       WRITE(6,8002)                                                G0    23
24  8001 FORMAT(10X,'SUSPECT FIRST STAGE EPILEPSY',//)                 G0    24
25  8002 FORMAT(10X,'SUSPECT SECOND STAGE EPILEPSY',//)                G0    25
26 *END G0                                                            G0    26
```

Fig. 10 Unprocessed Decision Table G0

```
 1       SUBROUTINE G0                                                G0     1
 2 C   DECISION TABLE FOR EPILEPSY                                    G0     2
 3 C                                                                  G0     3
 4       INTEGER  AND,A$(1),I$,N$,R$(1),T$SI,T$SY,V$,W$,X$            G0     4
 5       COMMON/NEUR1/CND(100),TUMOR(10),ISW(20)                      G0     5
 6 C                                                                  G0     6
 7 C        PART I    EVALUATE THOSE CONDITIONS WHICH REQUIRE         G0     7
 8 C                  A FIXED RESULT FOR ALL RULES.                   G0     8
 9 C                                                                  G0     9
10 C        CONDITION 1                                               G0    10
11 C                                                                  G0    11
12       IF (CND(37).EQ.1) GO TO 990                                  G0    12
13       GO TO 999                                                    G0    13
14   990 N$ = T$SI(R$,1,15)                                           G0    14
15       I$ = 1                                                       G0    15
16 C                                                                  G0    16
17 C        PART II   EVALUATE THOSE GROUPS OF CONDITIONS WHICH REQUIRE G0    17
18 C                  THE SAME RESULTS FOR ALL RULES.                 G0    18
19 C                                                                  G0    19
20 C        GROUP 1   CONDITIONS  2   3                              G0    20
21 C                                                                  G0    21
22       IF (CND(45).EQ.1) N$ = T$SY(R$,1 ,15)                        G0    22
23       W$ = N$                                                      G0    23
24       N$ = 1                                                       G0    24
25       IF (CND(46).EQ.1) N$ = 2                                     G0    25
26       IF(N$.NE.W$) GO TO 999                                       G0    26
27 C                                                                  G0    27
28 C        PART III  EVALUATE THE REMAINING TABLE USING AN           G0    28
29 C                  INTERRUPTED RULE MASKING TECHNIQUE.             G0    29
30 C                                                                  G0    30
31 C        CONDITION  4                                              G0    31
32 C                                                                  G0    32
```

Fig. 11 Processed Decision Table G0

```
      IF (CND(47).EQ.1) N$ = T$SY(R$,2 ,15)                          GO    33
C                                                                    GO    34
C        CONDITION  5                                                GO    35
C                                                                    GO    36
      IF (CND(48).EQ.1) N$ = T$SY(R$,3 ,15)                          GO    37
C                                                                    GO    38
C        MASK RULE 4              N N Y                              GO    39
C                                                                    GO    40
      IF( AND(R$(1), 37 ).NE. 37 ) GO TO 991                         GO    41
C                                                                    GO    42
C        EXECUTE ACTIONS          1                                  GO    43
C                                                                    GO    44
    1 A$(1) = 1                                                      GO    45
      GO TO 996                                                      GO    46
C                                                                    GO    47
C        MASK RULE 3              N N N                              GO    48
C                                                                    GO    49
  991 IF( AND(R$(1), 21 ).NE. 21 ) GO TO 993                         GO    50
C                                                                    GO    51
C        EXECUTE ACTIONS          2                                  GO    52
C                                                                    GO    53
  992 A$(1) = 2                                                      GO    54
      GO TO 996                                                      GO    55
C                                                                    GO    56
C        MASK RULE 2              Y N Y                              GO    57
C                                                                    GO    58
  993 IF( AND(R$(1), 38 ).EQ. 38 ) GO TO 992                         GO    59
C                                                                    GO    60
C        MASK RULE 1              Y Y N                              GO    61
C                                                                    GO    62
      IF( AND(R$(1), 26 ).EQ. 26 ) GO TO 1                           GO    63
      GO TO 999                                                      GO    64
C                                                                    GO    65
C        PART IV   EXECUTE THE PROPER SEQUENCE OF ACTIONS.           GO    66
C                                                                    GO    67
C        ACTION    1                                                 GO    68
C                                                                    GO    69
  994 WRITE(6,8001)                                                  GO    70
      GO TO 997                                                      GO    71
C                                                                    GO    72
C        ACTION    2                                                 GO    73
C                                                                    GO    74
  995 WRITE(6,8002)                                                  GO    75
 8001 FORMAT(10X,'SUSPECT FIRST STAGE EPILEPSY',//)                  GO    76
 8002 FORMAT(10X,'SUSPECT SECOND STAGE EPILEPSY',//)                 GO    77
      GO TO 997                                                      GO    78
C                                                                    GO    79
C        EXECUTE NEXT ACTION.                                        GO    80
C                                                                    GO    81
  996 V$ = A$(I$)                                                    GO    82
  997 IF(V$.LE.0) GO TO 999                                          GO    83
      W$ = V$ / 21                                                   GO    84
      X$ = V$ - 21 * W$                                              GO    85
      V$ = W$                                                        GO    86
      GO TO ( 994, 995 ), X$                                         GO    87
  999 RETURN                                                         GO    88
      END                                                            GO    89
```

Fig. 11 (continued)

6. THE THIRD SIMULATION TECHNIQUE

A DECISION TABLE INTERROGATION SIMULATION

The technique using decision table interrogation involves breaking a process into a basic set of actions. Each action is then considered from the point of view of causality. The various causal factors for each action are collected in the most convenient form, that of a decision table (with a single action entry). If each action is implemented in the form of a computer subroutine, and this subroutine is called as a result of "satisfying" the associated decision table, then iterative processing of the proper set of decision tables becomes a useful simulation technique. The technique is best explained by using an example. The example is that of the "duel-'til-death" problem detailed in Section 4. Instead of starting with the original problem statement, it is convenient to start with the ASTN formulation.

DECISION TABLE INTERROGATION

The conversion from an ASTN to a decision table interrogation simulation is very straightforward since the important result-to-cause viewpoint is already specified. The actual interrogation device can be the TNE subroutine, with possibly some additional code for setting the current values for the various state variables. Specific action routines, analogous to states in the ASTN, are designated. The cause for execution of each of these (simple) routines is specified in a minidecision table. Each decision table is analogous to the set of conditions specified on the arc(s) in the ASTM leading to (causing or allowing) the action-state. The entire set of conditions under consideration make up the components of an overall "state vector" that describes the current "state" of the process.

Decision tables must be set up carefully so that the action of one will not inhibit the querying of another; i.e., in Fig. 12 the action for decision table DA18 will stop the program unless either the action is to store a separate die event — which can be taken after all decision tables are queried. The problem is also addressed in Ref. 8, but is circumvented here by the lack of return lines (in Fig. 12) from A18 and A28.

Ref. 8. L. Gieszl, "Computer Program Modularization," APL/JHU TG-1223, September 1973.

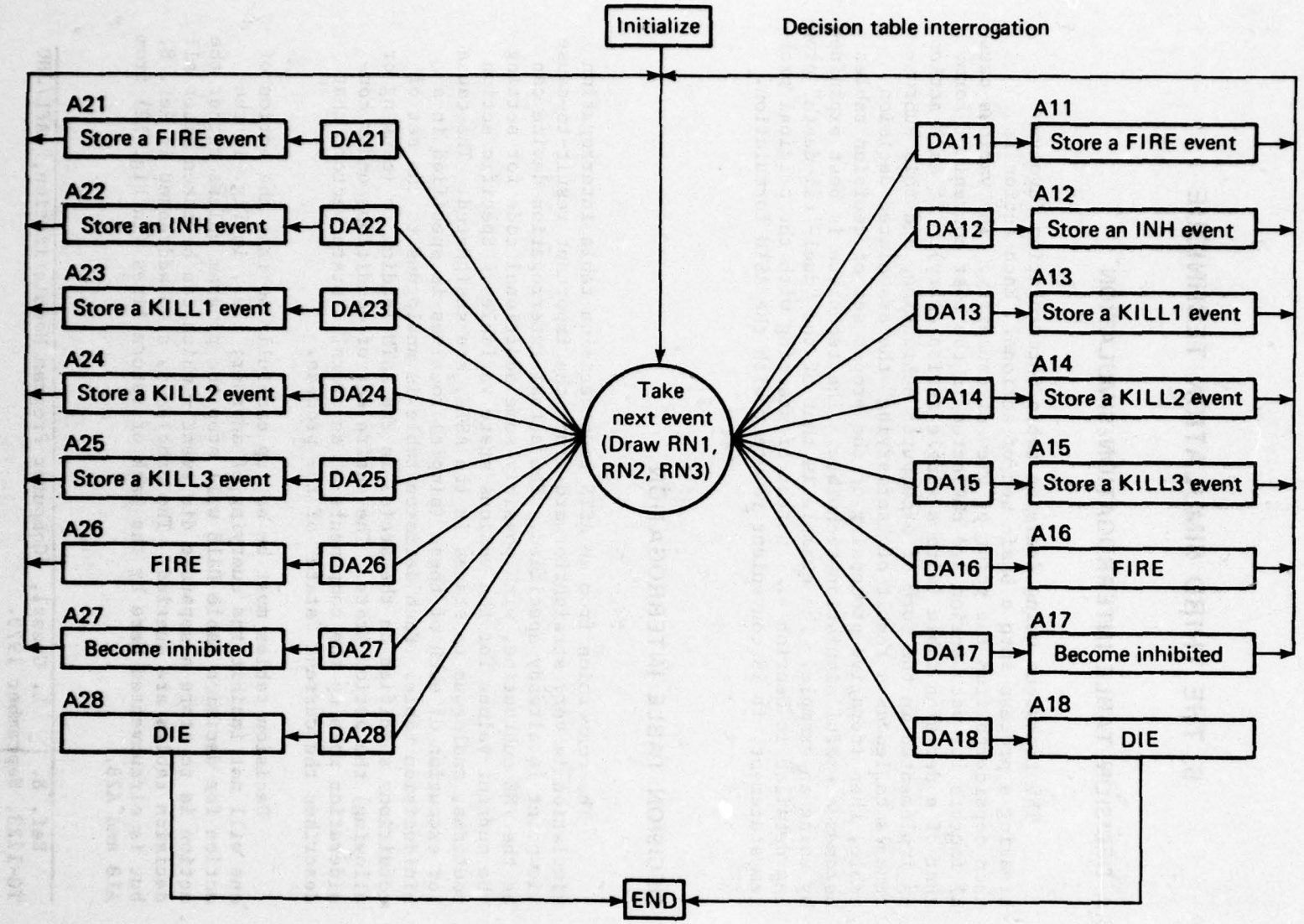


Fig. 12 Decision Table Interrogation for Simulation

Figure 12 is a fairly self-explanatory illustration of the decision table interrogation technique of programming. Figure 13 gives the list of corresponding decision tables in simple logical requirements instead of the formal decision table format desirable for more

DA11(DA21): Current event is not an INH or DIE

DA12(DA22): Current event is a FIRE event
$RN1 \leq P_H^1 \ (P_H^2)$
$P(3) < RN2 \leq P(4)$

DA13(DA23): Current event is a FIRE event
$RN1 \leq P_H^1 (P_H^2)$
$RN2 \leq P(1)$

DA14(DA24): Current event is a FIRE event
$RN1 \leq P_H^1 \ (P_H^2)$
$P(1) < RN2 \leq P(2)$

DA15(DA25): Current event is a FIRE event
$RN1 \leq P_H^1 \ (P_H^2)$
$P(2) \leq RN2 \leq P(3)$

DA16(DA26): Current event is a FIRE event

DA17(DA27): Current event is an INH

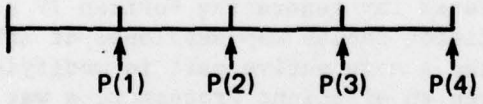


Fig. 13 List of Decision Tables in Simple Logical Requirements

complex examples. Finally, Table 1 gives a list of the action subroutines for this example. The descriptions of these routines show how simple the actual coded end-point actions should be in this type of formulation.

If a problem is so complex that, when it is modeled as a decision table interrogation simulation, the specified decision tables require formal decision tables (in contrast to the simple Boolean logic of Fig. 13), then a decision table preprocessor becomes an important factor in final implementation. The processor built at APL to convert

Table 1
List of the Action Subroutines for Example in Fig. 12

A11(A21): Store a FIRE event for T = CT + RN3 * 20 for 1 (2)

A12(A22): Store an INH event for T = CT + Δ4 for 2 (1)

A13(A23): Store a DIE event for T = CT + Δ1 for 2 (1)

A14(A24): Store a DIE event for T = CT + Δ2 for 2 (1)

A15(A25): Store a DIE event for T = CT + Δ3 for 2 (1)

A16(A26): Tally a fired shot

A17(A27): Erase any future FIRE event for 1 (2)

A18(A28): Record time of death, print out tallies, stop

decision tables to optimized Fortran code has made this method of simulation economically feasible (see Refs. 7 and 9).

A DECISION TABLE PROCESSOR

A processing system for generating Fortran IV source programs from limited entry decision tables was developed at APL. The system enables the user to take a more active part in modifying a computer-reliant war game. Given an efficient processor, a war game could be written in the form of decision tables and automatically converted to executable Fortran source routines. It is clear that modification time would be minimized , but only experience would show that actual simulation execution time was not significantly increased. The experience now gives us the confidence to term the system an "efficient processor."

Inputs to the processor consist of defined conditions, defined actions, decision table rules for relating action sequences with sets of condition results, and appropriate control cards. Outputs from the

---

Ref. 9. L. Gieszl, "Hierarchical Decision Tables, a Diagnostic Implementation," Proc. 1977 Autom. Control Conf., IEEE, 77CH1220-3CS, 1977.

processor consist of a list of inputs, a list of the resulting source program, and optionally, a data set of the generated program.

Each decision table program will consist of an initial section and a definition section defined by numbered processor control cards. The initial section contains the name of the subprogram, data and variable definitions, preliminary executable statements as required, and appropriate commentary.

The definition section contains defined conditions, defined actions, and decision table rules. There may be up to 30 conditions, 20 actions, and 50 rules.

Each condition, rule, and action is assigned an explicit number. At execution time for the resulting decision table subprogram, the rules are not generally tested in order, but based upon a weighting function generated by the decision table processor. Conditions are evaluated as required to permit the complete testing of a rule. Therefore, conditions are not generally evaluated in order of their respective number. The objective here is to reduce the overall execution time required of the resulting decision table subprograms.

The processor checks to ensure that each condition is defined mutually exclusive of the definition of every other condition, and that each action has some differences from all other actions.

Resulting decision table subprograms produced by this processor have their definition section divided into four parts. Part 1 evaluates the conditions that have either Y or N as their corresponding entries in all rules other than the else rule. If such a condition has all Y entries, for example, it would have to be evaluated true in order for any rule to be satisfied. Otherwise, it is known already that the else rule must be taken. Part 2 evaluates those groups of conditions that have the same set of Y and N corresponding entries in all rules other than the else rule. If one condition of such a group is evaluated as false, for example, all other conditions of that group must be evaluated as false in order for any rule to be satisfied. Otherwise, it is already known, that the else rule must be taken (on the first differently evaluated condition). Part 3 evaluates all other conditions and performs the rule-masking tests on rules. Part 4 is the properly coded executable sequence of actions. The satisfaction of any rule, other than the else rule, will result in a computed GO TO within Part 4.

The rule masking done in Part 3 is referred to as an interrupted rule masking technique. For example, in the next rule to be tested the processor determines whether or not all conditions, that have corresponding Y or N entries have been evaluated. If they have, then this rule may be tested. If they have not, the processor inserts

the proper Fortran coding at this point to evaluate those conditions and then inserts coding to test the rule. Note that conditions that have a corresponding hyphen (don't care) entry in a rule do not have to be evaluated prior to testing that rule. Hence, evaluation of conditions may be interrupted to test some rule(s) before all conditions are evaluated.

7. SUMMARY

The Event-Store simulation technique has been characterized as a procedure in which the designer determines some chronological order of events for each actor and then uses a queue handling executive (such as SNE/TNE) to allow for the apparent parallel processing of these sets of events. Event generation, regeneration, and appropriate actor interactions must be incorporated into the events themselves (e.g., FIRE in Fig. 3 includes storing another FIRE event for 2 + RN seconds from "now").

Both the Augmented State Transition Network and the Hierarchical Decision Table Interrogation techniques require an examination of the original problem from a "result-to-cause" perspective. In each case, all possible (state changing) activities are listed and carefully analyzed for possible causes. In the case of the ASTN, the causal chain for the various events determines the topology of the final simulation network. The Decision Table Interrogation technique maintains the autonomy of specific event causes. In this case, the set of all causal conditions for an event is the basis for a single-action decision table. The set of all appropriate decision tables is hierarchically interrogated (iteratively interrogated with the definition of the overall state vector relaxed with each iteration).

The technique involving decision tables is critically dependent on processing efficiency. Many tables may be interrogated thousands of times. The simple expedient of creating computer subroutines automatically from decision table formats, where the subroutines are optimized for fast exits, makes this technique practical. In other words, do not evaluate any more conditions than absolutely necessary for any given rule. Facilities are available in a decision table preprocessor that was implemented at the Laboratory.

INITIAL DISTRIBUTION EXTERNAL TO THE APPLIED PHYSICS LABORATORY*

The work reported in TG 1316 was done under Navy Contract N00017-72-C-4401. This work is related to Task Z02A, which is supported by Chief of Naval Operations, OP-96.

ORGANIZATION	LOCATION	ATTENTION	No. of Copies
DEPARTMENT OF DEFENSE			
DDC	Alexandria, VA		12
Industrial College of the Armed Forces	Washington, DC		1
National War College	Washington, DC		1
Department of the Navy			
CNO	Washington, DC	OP-96	1
		OP-604	1
		OP-941	1
NAVSEASYSCOM	Washington, DC	SEA-09G3	1
NAVAIRSYSCOM	Washington, DC	AIR-954	1
NAVELEXSYSCOM	Washington, DC	PME-108	1
NAVPRO/Laurel	Laurel, MD		1
Naval Air Dev. Ctr.	Johnsville, PA		
Naval Ocean Systems Ctr.	San Diego, CA	R. D. Becker	1
		H. Miller	1
Navy Post Graduate School	Monterey, CA		1
Department of the Army			
U. S. Army Research Institute	Arlington, VA		1
Department of the Air Force			
AFSC	Andrews AFB, MD		1
RADC	Griffis AFB, NY		1
U. S. GOVERNMENT AGENCIES			
Security Agencies			
CIA	Washington, DC		1
NSA	Ft. Mead, MD	R. N. Daniel, Jr.	1
National Aero. and Space Admin.			
Headquarters	Washington, DC		1
CONTRACTORS			
CACI	Arlington, VA	Dr. J. Fain	1
Martin Marietta Aerospace, Denver Div.	Denver, CO	J. Anino	1
The Rand Corp.	Santa Monica, CA	Hayes-Roth	1
G. E. TEMPO	Santa Barbara, CA		1
Sandia Labs.	Livermore, CA	C. Coll	1

Requests for copies of this report from DoD activities and contractors should be directed to DDC, Cameron Station, Alexandria, Virginia 22314 using DDC Form 1 and, if necessary, DDC Form 55.

*Initial distribution of this document within the Applied Physics Laboratory has been made in accordance with a list on file in the APL Technical Publications Group.