A DISCIPLINED APPROACH TO SOLVING PROBLEMS

by
E. H. Beitz (Spring - 1978)

Visiting Scholar, University of Utah, Salt Lake City, Utah, U.S.A.
Computer Systems Consultant, Saint Paul, Minnesota, U.S.A.

The process of designing a solution to a problem is difficult to describe. Prior experience, and a disciplined review of the pertinent data and relevant alternatives, can help to stimulate one's creative intuition. A mechanism for disciplining the reviewing process - in the form of decision tables - will be put forward. The same mechanism can be shown to be a useful guide to developing tests of the solution, enhancing and otherwise altering the system, and for communicating the intent of the solution to non data-processing people. All of the attributes of well-structured programs are easily retained and there are numerous possible alternatives for generating the system that carries out the solution.

The most important benefit to be derived from using a computer is the uniformity and discipline that it brings to a procedure. Programs invariably fail when situations that "can't happen" do, or when conditions that were not considered arise, (assuming, of course, that what was considered, was handled correctly). The methodology presented in this paper does not guarantee that a user will be successful because ultimately the discipline is self-imposed. Neither is it a sure-fire recipe for solving problems; it is rather a tool - a tool to help problem solvers organize their approach to implementing their solutions.

We all tackle a problem with a rag-bag of ideas and some *a priori* experience. Brinch Hansen (1973) is one of the few designers who honestly admit that the elegant algorithms that they publish are really the very highly refined products of a rather complex process. (A process which is sometimes difficult to understand!)

Parnas (1971) has suggested that the data be considered as the major element in determining how a solution is partitioned. While essentially agreeing with him, it is often difficult to decide on the set of data elements that will be required until a particular representation and / or algorithm is chosen.

A procedure may be treated as a finite state automaton. In order to model the finite state automaton one must make an initial partitioning - the set of states, the input alphabet, the output alphabet, the transition mapping from state to state, the starting state, and the final state, must all be chosen!

The methodology proposed below also requires an initial partitioning. The first attempt at partitioning may bear no resemblance to the final one; the important thing is that any partitioning that the designer feels might be important will suffice. An attempt to ensure that everything has been thought of will simply block the problem solving process.

INTRODUCTION TO THE METHODOLOGY

It is proposed that an unlimited entry decision table be used as the basic notation for representing the solution to a problem. The decision table may be thought of as an analog for a finite state automaton. To ensure the completeness of the set of states some constraints will be introduced into the process of determining the state of the automaton. The input alphabet will consist of a set of equivalence classes, each one of which is a sequence of actions that may or may not alter the environment. The environment comprises a set of data elements. The transition from one state to the next cannot be represented as a simple mapping but must be determined by examining the environment. During the process of determining the state it is vital that the environment is not altered! Each state implies the invocation of one, and only one, of the equivalence-classes in the input alphabet. The application of the sequence of actions of the equivalence class may alter the environment, which in turn will determine the next state, which in its turn implies the selection of the next sequence of actions to be executed. This cycle of events either does or does not terminate; but that depends on the nature of the problem.

CONDITIONS, RULES AND THE PROBLEM SPACE

The first step is to try to define the set of states that will completely cover the problem. This set of states will be called the problem space. (This too is an exercise in partitioning.) One or more conditions need to be decided on. What a condition is, is best illustrated by an example: Suppose that a 16-bit variable named **v** represents a positive integer in the range **x** to **x + 65535**; suppose too, that depending on which of a number of intervals **v** is in, the procedure that is to be carried out differs. The defined intervals are:

k, between **d** and **e** inclusive;

l, between **b** and **c** inclusive and between **f** and **g** inclusive;

and **m**, between **x** and **a** inclusive and between **c** and **d**.

These could be represented as a number line as follows:

Note that it is a true partitioning in that the intervals are disjoint. But there are some values that **v** could take for which no defined interval has been named! This discontinuous interval must be named in order to ensure completeness. Let us call it **n** and define it as:

n, when **v** is not in either **k** or **l** or **m**.

It isn't necessary to know all the details about a condition in order to start solving the problem. Neither do all the conditions have to be known at the outset. For each condition that is named it is only necessary to provide:

- a name - to identify the condition,
- an exhaustive set of alternatives - regardless of whether the partitioning is too fine or not fine enough,
- an environment - the set of variables that must be referred to in order to determine which alternative applies,
- and a brief description of what it is that the condition determines.

Let us call the set of alternatives for the condition C_1 the set CA_1. Now, if **m** conditions are defined then the problem space is defined by the Cartesian product of the sets of alternatives:

$$PS = CA_1 \times CA_2 \times \ldots \times CA_m.$$

Each state that is a member of the problem space is known as a rule. No matter how many new conditions or how many new alternatives are added to the problem space, it will always be possible to identify exactly which rules are effected! This means that those parts of the solution that have already been dealt with will in most cases not be altered when the new situations are discovered. The problem space is combinatorially complete and even when the conditions or alternatives are permuted every rule will still be present.

It's important that a strict enumeration of the rules be possible. To make this possible it is necessary to order each set of alternatives. This is simply achieved by mapping each set of alternatives onto a zero-based index set. This serves to give the resulting decision table representation of the solution a desirable regularity, which in turn makes it possible to recognize patterns by eye-balling the table.

ACTIONS AND EQUIVALENCE CLASSES

Before considering how the actions are related to the conditions and rules let us look at how an action is defined. In a manner similar to the conditions it is necessary to provide some information for each of the actions, as follows:

- a name - to identify the action,
- a reference environment - the set of variables that are read by the procedure that implements the action,
- an affected environment - the set of variables that might be altered by the procedure that implements the action,
- a brief description of what the action accomplishes, and
- the set of equivalence classes in which the action is included.

Each rule will require that a specific sequence of actions be executed. The similarity of the sequences for a number of different rules is precisely what makes the computer so usable. One of these action sequences defines an equivalence class, and the equivalence class itself is represented by two sets: the action sequence defining the equivalence class, and the set of rules for which that action sequence is invoked.

These action sequences may not in fact be strict sequences! Provided that two actions are not sequentially dependent and that the entire environment of each is disjoint with respect of the others affected environment, there is no reason why both of them cannot be executed concurrently. A notational device, that allows this concurrency within an equivalence class to be represented, will be found in Appendix A.

THE PROBLEM SOLVING PROCESS

In an unpublished working paper (Beitz 1974) I proposed that an interactive system for solving problems represented in decision table form, be implemented. The method may best be summarized as an attempt to permute the conditions, the sets of alternatives and the actions so that those which are similar are adjacent to one another. The problem space for most solutions is usually too large to represent on a single piece of paper in its complete form. However, after reducing the table by successive permutations of its constituents, most problem solutions seem to fit on a sheet of letter-sized paper (without having to resort to very small printing).

An example of a hypothetical problem solution will be presented below; first in its complete form and then in what might be called its final form. We will assume that the problem space is defined by the three sets of alternatives CA_1, CA_2 and CA_3:

$$CA_1 = \langle d,\ e,\ f,\ x = \overline{d \text{ or } e \text{ or } f} \rangle$$

$$CA_2 = \langle h,\ j,\ k = \overline{h \text{ or } j} \rangle$$

$$CA = \langle g,\ \overline{g} \rangle$$

The resulting problem space will look like this:

C_1	d						e						f						x					
C_2	h		j		k		h		j		k		h		j		k		h		j		k	
C_3	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$
RULE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Suppose that the set of actions over which the equivalence classes are defined is:

$$\{A_1, A_2, A_3, A_4, A_5, A_6\}$$

Let us also assume that whenever both A_2 and A_4 are members of the same equivalence class, A_2 must be completed before A_4 is begun; expressed as a constraint as follows:

$$<A_2, A_4>$$

The complete initial table might be:

C_1	d						e						f						x					
C_2	h		j		k		h		j		k		h		j		k		h		j		k	
C_3	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$
RULE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
A_1	✔			✔			✔	✔		✔				✔		✔				✔		✔		
A_2	✔	✔	1				✔		1				1	✔	1				1	1	1			
A_3		✔	✔	✔					✔	✔				✔	✔	✔					✔	✔		
A_4			2	✔				✔	2	✔			2		2	✔			2	2	2	✔		
A_5	✔			✔			✔	✔		✔				✔		✔				✔		✔		
A_6					✔	✔					✔	✔					✔	✔					✔	✔
EC	0	1	2	3	4	4	0	5	2	3	4	4	6	7	2	3	4	4	6	8	2	3	4	4

The process of reducing the table may now be described. The designer might realize while dealing with a specific rule that some refinement is possible. For example, suppose that the rules were presented to the designer in the order of the complete table

(above). When rule **11** is presented the designer realizes that whenever C_2 takes alternative **k** then action A_6 is executed, and what is more, is the only action that is executed. This may be verified for those rules already dealt with - namely for rules **4**, **5** and **10**. There are only two possibilities: either the designer's insight is substantiated by this check or it is refuted! (The confirmation does not necessarily imply that it is 'correct'; this decision is the designer's. Refutation, on the other hand, is far more significant!) If the hypothesis is refuted then it becomes necessary to examine the contradiction. In the particular case presented in this example, if the designer decides that the insight was indeed correct, then it will no longer be necessary to consider rules **16**, **17**, **22** and **23**. These rules are essentially done.

Situations like the one just described are the rule rather than the exception. The eight rules that are affected by the acceptance of the hypothesis in our example may be collected together. This is done by changing the order of the conditions. Simply making C_2 the primary condition at the head of the table will suffice to make the eight rules adjacent. Any change to either the order of the conditions or the order of the alternatives in a single set of alternatives, will change the order of the rules. To conform to the strict enumeration the rules may simply be renamed.

There will be a one-to-one onto mapping from the old rule names to the new rule names. The following problem space is identical to the one in our example above and both the old and the new rule names are shown:

C_2	h								k								j							
C_1	x		e		f		d		x		e		f		d		x		e		f		d	
C_3	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$	g	$\bar{g}$
OLD RULE	18	19	6	7	12	13	0	1	22	23	10	11	16	17	4	5	20	21	8	9	14	15	2	3
NEW RULE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

After permuting the conditions of our example the final table for the solution to the problem might look like the table on the following page.

It is quite evident that this final table is a lot simpler to comprehend than the complete table. The actions and the conditions and their alternatives may be at as high a level of abstraction as the designer wishes. Each action or condition may be represented by a decision table in its own right. This process will be known as elaboration. There is an important difference between the table representing the elaboration of a condition and that of an action. The condition's table may not include any actions! The elaboration of a condition serves to do nothing more than select an alternative. In the case of an action

the elaboration is a decision table similar to that of the problem itself with the constraint that the environment of the action and its elaboration are identical.

C_2	h						j		k
C_3	g		$\overline{g}$				g	$\overline{g}$	$p \in CA_3$
C_1	d or e	f or x	d	e	f	x	$r \in CA_1$	$r \in CA_1$	$r \in CA_1$
RULE	0, 1	2, 3	4	5	6	7	8..11	12..15	16..23
A_1	✔			✔	✔	✔		✔	
A_5	✔			✔	✔	✔		✔	
A_2	✔	1	✔		✔	1	1		
A_3			✔		✔		✔	✔	
A_4	✔	2		✔		2	2	✔	
A_6									✔
EC	0	6	1	5	7	8	2	3	4

When a table invokes an action whose elaboration is that table itself, we have recursion. The important aspect of the whole scheme is that the problem is treated as a unified object and is essentially a nested hierarchy of sub-problems. Any one of a number of different programs may be coded to represent the solution and testing also becomes an orderly procedure. Ultimately as was pointed out in the beginning, the discipline is self-imposed but the tables make it much easier for the designer to live with this imposition!

APPENDIX A

A NOTATIONAL DEVICE FOR REPRESENTING EQUIVALENCE CLASSES

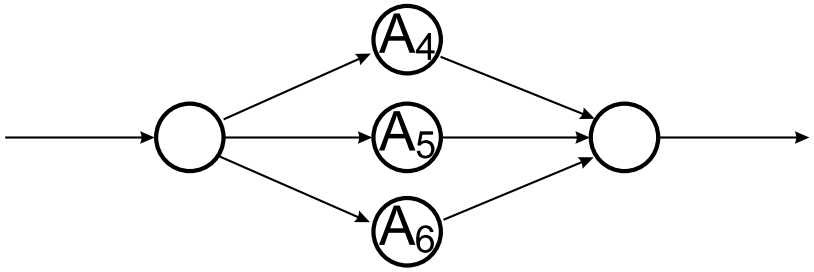
A sequence of actions may sometimes have some strict order in which the individual actions must be executed. This is usually required when more than one action refers to the identical data environment. The designer must specify the order in which such actions are to be executed. The set of action names, delimited by commas, and enclosed in brackets of one form (say **<** and **>**) would indicate that those actions must be executed in sequence starting with the leftmost. When actions may be executed concurrently they will be enclosed in brackets of another form (say **{** and **}**). A single element in a sequenced action set may be a set of concurrent actions and similarly a single element in a set of concurrent actions may be a sequence of actions.

For example, if the action sequence for an equivalence class consists of A_1 followed by A_2 followed by A_3 then the action sequence for the equivalence class is:

$$<A_1,A_2,A_3>$$

or, graphically:

An example of a concurrent set of actions might be (graphically):



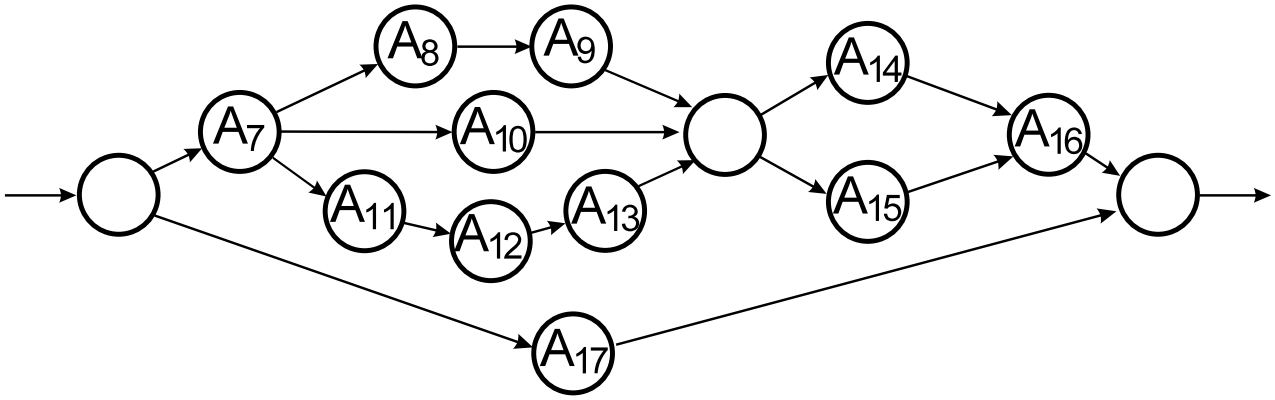
or, in our notation:

$$\{A_4,A_5,A_6\}$$

which is identical to all of these:

$$\{A_4,A_6,A_5\}\ \{A_5,A_4,A_6\}\ \{A_5,A_6,A_4\}\ \{A_6,A_4,A_5\}\ \{A_6,A_5,A_4\}$$

A more complex example combining both forms might be (graphically):



or using the notation:

$$\{A_{17},<A_7,\{A_{10},<A_8,A_9>,<A_{11},A_{12},A_{13}>\},\{A_{14},A_{15}\},A_{16}>\}$$

BIBLIOGRAPHY

Beitz, E. H., ***A proposed method for interactive problem solution using an exhaustive, combinatoric, decision-table technique.*** Working paper - Decision Table Task Group of CODASYL, 1974.

Brinch Hansen, P., ***Operating System Principles.*** Prentice-Hall, 1973.

Parnas, D. L., ***Information distribution aspects of design methodology.*** In: Proceedings of IFIP Congress 1971, August, 1971. North Holland, 1972.

The author was invited to present this paper at a conference at the University of Aarhus in Aarhus, Denmark in May 1978.