WORKING NOTE AC47

Alan Creak
5 March 1986

REPRESENTATIONS FOR COMPOUND STATEMENTS

(This note preserves in more accessible form, and with minor amendments for clarification, a document which I first wrote on 24 October 1970.)

NOTE ON POSITIONAL VERSUS LIST LOGIC PROCESSORS.

The logic processing required in the system is basically a set of operations on various compound statements which involve elemental statements A, B, C.... The compound statements are manipulated in various ways to produce results which are also in terms of the elemental statements, and some discussion of the most convenient realisation of these processes on the computer is in place.

Two methods suggest themselves : a list method, corresponding closely to the "pen-and-paper" methods of symbolic logic; and a method involving positional notation to distinguish the elementa1 statements. These methods will first be outlined.

THE LIST METHOD.

Each elemental statement is associated with two symbols : A and -A, for example. A compound statement is composed of several elemental statements and appropriate Boolean connectives & and v. (We avoid the use of brackets throughout, in the hope of avoiding unnecessary complication. This undoubtedly leads to some inefficiency in the use of storage space – thus, the statement A v (B & C) must be replaced by the conjoint assertion of the two compound statements A v B and A v C – but is accepted in the interests of simpler programming.) In the "list" method, each compound statement is stored as a character string, much as it would be written on paper. The important characteristic of this mode of storage, for present purposes, is that each statement included must be explicitly specified by some suitable coding method. At the same time, elemental statements not involved in the current compound statement need not be included.

THE POSITIONAL NOTATION.

Each compound statement is represented by a vector, with one element for each elemental statement. As each elemental statement is distinguished only by its position in the vector, every compound statement must include space for every elemental statement. An elemental statement can therefore now be associated with three different symbols : A, -A, and (A), say, where the last denotes the absence of A from the compound statement. The points in favour of this notation are that, as the symbol is denoted by the position in the vector, it need not be specified explicitly, and, in consequence, many statements can be parsed into a single computer word, thus saving processing time by allowing several elemental statements to be processed simultaneously. Several different ways of coding the statements in detail can be devised; each (short of extensive coding and decoding programmes) will use two bits for each statement. The scheme adopted for this programme is to denote assertions

and denials of individual elemental statements by 1 bits in separate vectors, all other bits being set to zero; assertion, denial, and non-involvement are thus represented by the bit pairs (1,0), (0,1), and (0,0); the fourth possibility, (l,l), is a contradiction, and its detection is used to initiate deletion of the compound statement in which it is found.

The positional notation was chosen for use in this work mainly for its simplicity from the programming point of view, and for its economy of storage. To demonstrate the economy achieved, consider a fairly typical case involving 20 elemental statements. (This number is deliberately chosen to exceed the number of bits in a computer word; 16 statements is an especially favourable case for the positional notation.) We will suppose, again typically, that a compound statement contains 3 elemental statements; as compound statements are combined together in the course of the processing, new statements are produced which may finally involve all 20 original statements, and we will also consider the case of compound statements containing 15 elemental statements.

We note first that the minimum storage required to provide separate codes for 20 statements is 5 bits, and, as each statement can be asserted or denied, 6 bits will be required altogether; it follows that, unless the codes are allowed to spread from one computer word to the next, which would introduce an extra complication into the programming, two elemental statements can be accommodated in each computer word. (Observe that a restriction to 16 elemental statements would be favourable to the list method too, as it would allow three statements to be packed into a single word; on the other hand, this would involve either special programming for systems of 16 or fewer statements, or restriction of the application of the programmes to logical systems with fewer than 17 statements, and neither of these alternatives was thought acceptable.) The compound statement with three elements will therefore occupy two computer words, that with 15 statements, 8 words. (The mild deception apparently introduced by using odd numbers is only apparent; any string of this sort would necessarily require a count of elements, and this could hardly occupy less than a half-word without introducing either further restrictions or programming complications.)

For the positional notation, on the other hand, each compound statement, no matter how many elemental statements are involved, requires four words of storage (again eschewing any packing economies). There is thus an advantage to the list method for the shorter compound statements, and to the positional notation for the longer ones. The balance is turned markedly in favour of the positional notation by the observation that, in actual calculations, the bulk of the storage difficulties arise towards the end of the process, when the statements being handled already contain references to most of the elemental statements; and, finally, that the programming is very much easier – and certainly very much faster in execution – for the positional notation.