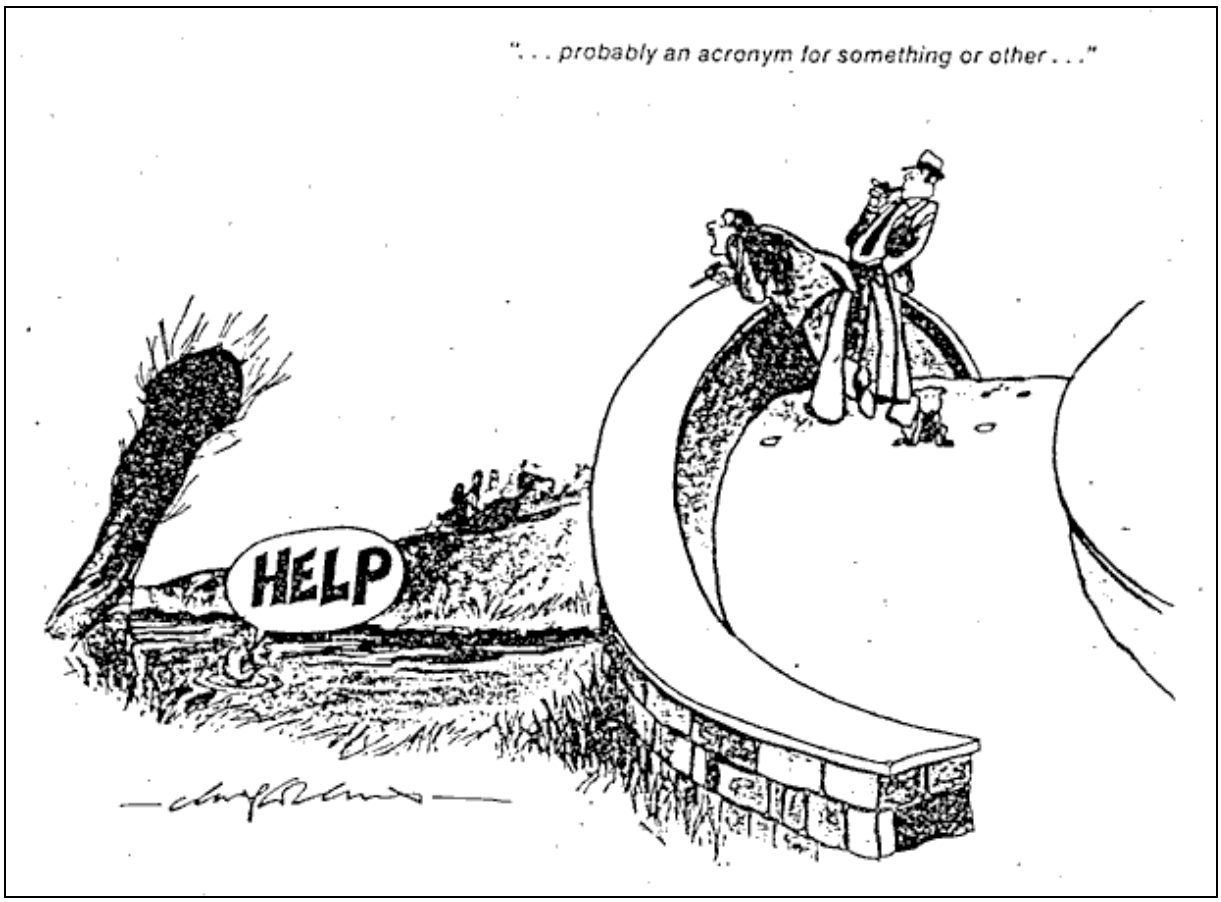
WORKING NOTE AC53

Alan Creak
28 May 1986

EVALUATING EXPRESSIONS IN ECCLES.



Here's the Eccles Logic Processor

(I've been saving that for years ! [5])

One of the problems with the Eccles way of evaluating expressions is combinatorial explosion. To work out the disjunctive normal form expression corresponding to the conjoined assertion of several statements, one must somehow evaluate all possible "cross-products" of the terms in the individual statements, so that a very large number of terms may be produced. The problem isn't new, of course : it's mentioned by Chang and Lee [1] (interestingly, with conjunction and disjunction interchanged) as the "multiplication method"; they also give some "rules" which purport to avoid the explosion. Unfortunately, these rules are no good to Eccles, as they are directed simply at demonstrating the unsatisfiability of a set of clauses; Eccles needs to finish up with the set of solutions, if there is one, so there's no simple way to translate the Chang and Lee rules to an Eccles equivalent.

NOTICE

> *I am also in trouble with a terminological explosion. I have statements, terms, assertions, simple statements, composite statements … - and I've no idea which means what, or when to use each. I think I've got atoms right; literals (which I rarely seem to talk about) are atoms or negated atoms; clauses are* ***disjunctions*** *of literals, so probably inappropriate for Eccles. Help ?*

Nevertheless, not all is necessarily lost. While we can't in general avoid the explosion completely, we can at least hope to control it a little, and we can even in principle cope with it if it happens.

CONTROLLING THE EXPLOSION.

The order in which we incorporate the initial statements into the final result is immaterial; so we can choose an order which looks as if it might muffle the explosion as much as possible. The obvious way of doing so is to choose terms which lead to as much cancellation of terms (that is, appearance of **false** terms which we can omit) as possible. Therefore, if we find that an atom X is frequently asserted in the composite statement, we look for a statement in which X is frequently denied, and combine that next. (There may be other ways, but I think this is the obvious one.) It's worth observing that this has something in common with the resolution method [4], with the difference that we carry on all the arguments in parallel rather than following a single thread.

This one certainly works. Here are some measurements I made on my ancient IBM1130 version of Eccles, where the explosion really was a problem [2].

The experiments were on the chemical reaction mechanism system (which I briefly described [3], but really should do properly some day). The direct input to the Eccles part was the disjunctive form bit array representation, which was not congenial for manual use, so I used an indirect method throughout - I entered data as chemical reaction equations, and left the "front end" software convert that into the Eccles representation. The equivalent of the "cancellation of terms" was, reasonably enough, a sort of "cancellation of chemicals" - roughly, making sure that substances which had appeared as reactants also appeared as products. (In fact, it worked by making statements alternately about reactions and about compounds, but that's what it amounted to.)

IBM1130 Eccles carried only a single statement, which was the conclusion so far : individual statements were not stored. The statement was stored in an array which could accommodate a few hundred simple statements; as the old and new statements were both held in the array (one at each end) during a calculation, congestion was acute. I "solved" the explosion problem very simply : when the array became full I stopped evaluating. (This is a rather crude version of the "store part of the intermediate expression while you finish off the calculation on the rest" idea, described below - though as I didn't actually store any of it, big calculations tended to finish up with all terms eliminated.)

For the relevant experiments, I compared the performance of IBM1130 Eccles on working out reaction mechanisms giving the same set of reactants in two different orders - one random, and one ordered to encourage cancellation where possible. The result was, in one way at least, as expected : the order of presentation of the reactions had a profound effect on the performance of the system. For the first try, the numbers of simple statements in the compound statement after each step for the two methods were :

Random order :	2	3	3	5	13	13	26	39	104	104
"Optimum" order :	2	5	2	4	6	6	8	14	8	8
Random order :	169	133	165	133	164	133	171	133	171	77
"Optimum" order :	8	9	11	20	9	7	14	14	22	22
Random order :	5	5	8	3	1	1	2	2	0	-
"Optimum" order :	44	44	76	76	140	133	176	133	177	133

There are reasons for believing that neither result is perfect (the "170 - 130" alternation is symptomatic of a full array, with room for about 300 elementary statements), but clearly the ordered calculation takes much longer to fill up the array, and does eventually arrive at a result; the random calculation doesn't. The order of evaluation does indeed make a big difference to the behaviour of the growing composite statement. It is unfortunate that the next experiment gave exactly the opposite result ! - but the strong effect of order of evaluation on the explosion remains clear.

COPING WITH THE EXPLOSION.

If even with the best avoidance strategy which we can find the explosion is still serious, there is a way round. (Though one could wonder if a serious explosion might not mean that the question itself is somehow silly : are we really likely to be interested in problems with thousands of solutions ?) This detour is based on the observation that we don't actually *need* to do all the evaluations in parallel; if it's convenient, we can do some in series. Indeed, at any point in the evaluation, we have a composite statement C, say, which must be combined with further statements S_1, S_2, … to give the answer C & S_1 & S_2 & …. We are equally entitled to express C as (C_1 v C_2), and to perform the evaluation as (C_1 & S_1 & S_2 & …) v (C_2 & S_1 & S_2 & …). To do that, we would remember - perhaps as a disc file - part of C and the remaining statements S_i, complete the evaluation with the rest of C, store the result, retrieve the stored fragment of C, complete the calculation on that, and finally combine together the two partial results.

This wouldn't solve all our problems - it wouldn't cope with a really serious explosion - but it would be feasible as a way of increasing the effective storage by a factor of 10 or so.

RELEVANCE TO NEGATION.

The same comments apply, more or less, to negation. We can illustrate the phenomenon using the statement :

```
{ ( a & b & c ) v ( d & e ) v ( f & g & h ) }          (𝔸)
```

Negated, that becomes :

```
~ { ( a & b & c ) v ( d & e ) v ( f & g & h ) }        (𝔹)

= ~( a & b & c ) & ~( d & e ) & ~( f & g & h )

= { ~(a) v ~(b) v ~(c) } & { ~(d) v ~(e) } & { ~(f) v ~(g) v ~(h) }
  (ℂ)
```

The inner brackets in the final statement ℂ emphasise that, in Eccles terms, each of the individual atoms occupies a statement of its own. The result obtained in the next step contains (assuming no cancellation) three statements after the first term, six after the second, 18 after the third, and so on.

But the point is that statement ℂ is a "conjoined assertion of several statements" - which is exactly the same operation as we had before, and the same arguments apply. This time, though, it's perhaps a little easier, because the initial expression 𝔸 presents us with a lot of information about the statements to be conjoined in a rather compact form (so the assertion `( d & e )` in the original 𝔸 shows that we shall have a statement `( ~(d) v ~(e) )` in the final ℂ), and each component of a statement (such as `(a)`) contains only a single assertion or denial. It follows that the original statement is a direct representation of the final, provided that we use an appropriate interpretation - so one way of representing the result would be simply to note that the original representation was to be interpreted in a different way !

This is not really adequate, though, because sometime we are going to have to do the calculation if we are to get the answer we want. (It may nevertheless be worth considering as an intermediate representation, which we only actually evaluate when we're forced to - see below.) So we return to the idea that evaluating the negation operator is in essence the same as evaluating the conjunction of a bunch of statements, and that therefore we can use the same techniques to control the possible explosion : we can select the order of combination of the statements - and this is made easier, because we now have a very simple way of checking for the occurrence of suitably negated atoms in the component terms; or we can perform the evaluation in parts.

There is one further possibility, which may be applicable in certain circumstances. If the statement is being negated in a context where it will immediately be conjoined with other statements, then it could be convenient to stop the negation at the last point of the example above (corresponding to $\mathbb{C}$), and then to include all the component statements simply as additional terms to be conjoined with the others. In this case, it might well be advantageous to regard the original statement as a separate notation for the final set, in accordance with the suggestion made earlier.

REFERENCES.

(1) C-L. Chang, R.C-T. Lee : *Symbolic logic and mechanical theorem proving*, Academic press (1973), p63.

(2) G.A. Creak : Experiment notes, 7 September 1972.

(3) G.A. Creak : *ECCLES*, Working note AC50, 12 May 1986.

(4) E. Rich : *Artificial intelligence* (McGraw-Hill, 1983), p153.

(5) New Zealand Listener **91 #2044**, 10 March 1979.