WORKING NOTE AC51

Alan Creak
12 May 1986

ECCLES AND ELIZA.

WHAT'S WRONG WITH ECCLES ?

A major disadvantage of Eccles [1] is that it is restricted to zero-order (propositional) logic, in which it is not possible to make propositions about variables. Thus, Eccles can handle arguments of this sort :

Daisy is a cow ;
If Daisy is a cow, then Daisy eats grass;
Therefore Daisy eats grass.

But Eccles must be given each of those assertions explicitly. It cannot infer the second from the general statement "All cows eat grass" – or, equivalently,

For any object X : If X is a cow, then X eats grass.

This does not, of course, detract from Eccles's utility in circumstances where it can be used; but it does limit those circumstances to finite (and, in practical cases, quite small) collections of assertions; because, for any even moderately large set of objects, we wish to make general statements like "All cows eat grass", rather than provide a separate statement for every cow in our universe.

Logic languages like Prolog are based on the first order (predicate) calculus, in which universal statements can be made, and the Prolog system is designed to apply general statements to special conditions by matching constants to variables. Thus, given the two statements

Cow(Daisy)
and Cow(X) –> eats(grass, X)

the Prolog system can deduce that

eats(grass, Daisy)

WHERE DOES ELIZA COME IN ?

Eliza is the name of a computer programme originally written many years ago by Joseph Weizenbaum [2], which is designed to carry on a conversation. It is an accepted fact that Eliza is not an intelligent programme [3]; all it does is compare its input with a lot of patterns which it keeps in its memory until it finds a matching pattern, then it constructs a response by transforming the text of the input statement according to a rule associated with the matched pattern. (It is tempting to ask whether Expert Systems, which everyone knows to be excellent examples of artificial intelligence [4], do much more.) For example : an input sentence

Tomorrow I am going to Wellington

might be matched by a pattern

Tomorrow I am going to X

with the associated response pattern

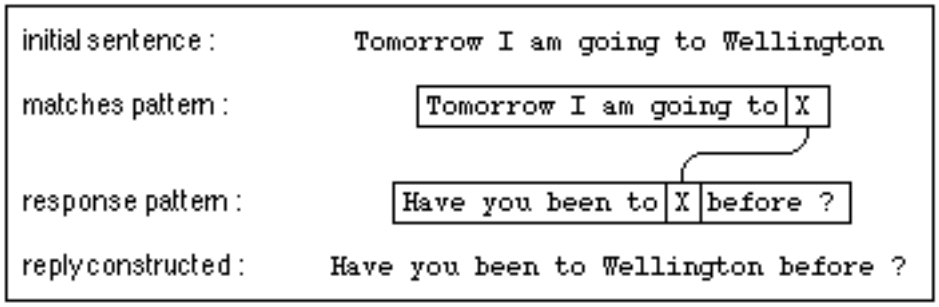
Have you been to X before ?

leading to the reply

Have you been to Wellington before ?

If the input sentence made sense, then – provided that the matching operation is carefully designed – the output will make sense too; and an Eliza with a large number of such patterns and transformation rules can keep up a very impressive, though ultimately quite pointless, conversation. The proviso in that sentence is critical; the patterns used, and the order in which they are tried, must be meticulously tailored if the replies produced are to remain credible. What would happen to the example given if the initial sentence had been "Tomorrow I am going to scream" ?

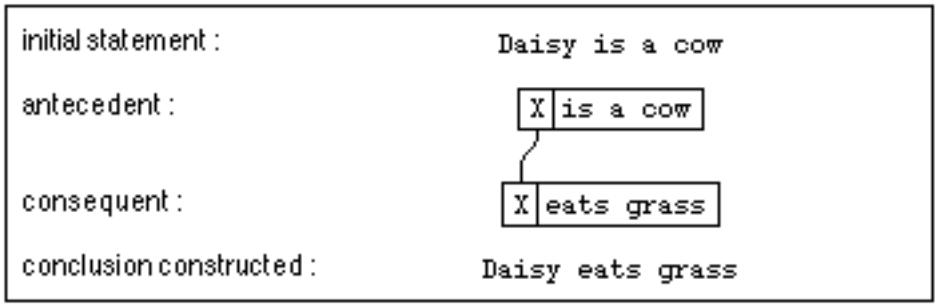
Here's a picture of Eliza's basic operation :



Remember it.

WHAT'S THAT GOT TO DO WITH ECCLES ?

What do we need to make Eccles work like (or, for preference, better than) Prolog ? We need the capacity to generalise. We need, in fact, to perform a transformation like this :



That should look familiar. It does ? Good ! I can sum up all that in this inequality:

Eccles + Eliza > Prolog

Why do I assert that the combination of Eccles and Eliza will exceed Prolog in some way ? Because Eccles can handle negation precisely, and Prolog can't. Further, Eliza-Eccles should be able to cope with reasoning in higher-order logic in a more natural way than Prolog [(5)], which has to do it by defining "meta operations" of various sorts.

This is how it will (perhaps) work.

First order reasoning :

1. We tell Eccles :

   X is a cow –> X eats grass.

2. We tell Eccles :

   Daisy is a cow.

3. Eccles looks for terms including "Daisy is a cow" with which it can combine the new information. (I suppose it should have done the same for "X is a cow" etc, but we'll suppose nothing happened.)

4. Eccles now tries to match "Daisy is a cow" with appropriate items in its symbol table, by instantiating variables. (By simple matching ? or maybe something like unification ?). It infers :

Daisy eats grass.

Or something along those lines. Notice that the pattern matching is quite distinct from the reasoning; the reasoning is performed with bit arrays, as described earlier, but the pattern matching is done on textual items in the name table. Notice, too, that the pattern matching can work in either direction; as with Prolog, the process can be driven by an attempt to work backwards from "Daisy eats grass" rather than forwards from "Daisy is a cow".

Higher order reasoning :

To show how Eliza-Eccles might deal with a problem involving higher-order reasoning, consider the statement : *If X is honest, and X says Y, then Y is true*. That statement involves higher-order reasoning, because it makes an assertion *about* an assertion : that statement Y is true. You can do it in Prolog, but it's certainly outside the ordinary realm of straightforward logic programming. Eliza-Eccles, on the other hand, reasons like this :

Given that :

(X is honest) => ((X says Y) => Y).
Albert is honest.
Albert says Mary was there.

we first combine the first two to obtain :

(Albert says X) => X.

then combine this with the third statement to derive :

Mary was there.

The magic, if there is any, perhaps comes from the separation of the reasoning and substitution steps : the pattern matching doesn't concern itself with the logic of the statements, only the textual form; while the reasoning ignores the text and concerns itself only with the logic.

Once we have something that'll do all that, then we can start to experiment with it. We could also start to explore its theoretical power, but I don't know how to start. (Note that one useful thing about logic programming in predicate logic is that we have a good knowledge of the mathematical properties of the notation we use : I don't know just how important that might or might not be.) I'm just not sure what its capabilities will be – but they should be interesting.

NOTATION.

It may turn out to be convenient to formalise the way in which statements are represented, perhaps to something like the conventional predicate logic pattern. This makes the pattern matching much less touchy, and helps to avoid the ambiguities of English parsing which lead to confusing similarities between sentences including "Wellington" and "scream", as illustrated earlier. I would prefer to avoid such disciplines for as long as we can, though, because I think they can constrain one's thinking : one tends to think in terms which are easy in the language one is using, and a constrained language can easily lead to constrained thought. If we keep the notation as free as possible for as long as possible, we're more likely to try to say hard things, a therefore be brought up against challenging problems.

It's also worth pointing out that a formal notation isn't the only way to resolve ambiguity : an alternative is to make the input end of the programme clever enough to distinguish "Wellington" from "scream". I know that's not easy either, but it's certainly interesting; and one nice thing about Eliza-Eccles's separation of reasoning from parsing is that we don't have to choose a notation that lends itself easily to logical manipulation.

REFERENCES.

(1) G.A. Creak : *ECCLES* , Working note AC50 (1986).

(2) J. Weizenbaum : *ELIZA – a computer program for the study of natural language communication between man and machine*, Comm.ACM **9**, 36 (1966).

(3) E. Rich : *Artificial intelligence* (McGraw-Hill, 1983) p305 (and see also reference 2).

(4) E. Rich : *Artificial intelligence* (McGraw-Hill, 1983) p284.

(5) W.F. Clocksin, C.S. Mellish : *Programmimg in Prolog* (Springer-Verlag, 2nd edition, 1984) p255.