# ARTIFICIAL INTELLIGENCE

J McCARTHY

**Rapporteur:** John E Dobson

*I meant what I said, and I said what I meant.*
*An elephant's faithful, one hundred percent!*
moreover,
*An elephant never forgets!*

# Elephant 2000: A Programming Language Based on Speech Acts

**by John McCarthy, Stanford University**

Abstract: Elephant 2000 is a vehicle for some ideas about programming language features. We expect these features to be valuable in writing and verifying programs that interact with people (e.g. transaction processing) or interact with programs belonging to other organizations (e.g. electronic data interchange)

1. Communication inputs and outputs are in an I-O language whose sentences are meaningful speech acts approximately in the sense of philosophers and linguists. These include questions, answers, offers, acceptances, declinations, requests, permissions and promises.

2. The correctness of programs is partially defined in terms of proper performance of the speech acts. Answers should be truthful and responsive, and promises should be kept. Sentences of logic expressing these forms of correctness can be generated automatically from the form of the program.

3. Elephant source programs may not need data structures, because they can refer directly to the past. Thus a program can say that an airline passenger has a reservation if he has made one and hasn't cancelled it.

4. Elephant programs themselves will be represented as sentences of logic. Their properties follow from this representation without an intervening theory of programming or anything like Hoare axioms.

5. Elephant programs that interact non-trivially with the outside world can have both *input-output specifications*, relating the programs inputs and outputs, and *accomplishment specifications* concerning what the program accomplishes in the world. These concepts are respectively generalizations of the philosophers' *illocutionary* and *perlocutionary* speech acts.

6. Programs that engage in commercial transactions assume obligations on behalf of their owners in exchange for obligations assumed by other entities. It may be part of the specifications of an Elephant 2000 programs that these obligations are exchanged as intended, and this too can be expressed by a logical sentence.

## Examples

- Speech acts

  Requests (authorized, **comprehensible**)

  Questions (comprehensible)

  Answers to questions (**truthful and responsive**)

  Offers (authorized)

  Acceptances and refusals

  Promises (authorized and **kept**)

- Reference to the past

  A passenger has a reservation **if he has made** one and hasn't cancelled.

# ELEPHANT 2000 AIRLINE RESERVATION PROGRAM

**if** $\neg$ *full flt* **then accept.request make commitment** $admit(psgr, flt)$

**answer.query exists commitment** $admit(psgr, flt)$

**accept.request cancel commitment** $admit(psgr, flt)$

**if now** $=$ **time** *flt*

$\wedge$ **exists commitment** $admit(psgr, flt)$

$\wedge \neg$ *full1 flt*

**then accept.request** $admit(psgr, flt)$

*full flt* $\equiv$

$card\{psgr \mid$ **exists commitment** $admit(psgr, flt)\} = capacity\ flt$

*full1 flt* $\equiv$

$card\{psgr \mid$ **did** $admit(psgr, flt)\} = capacity\ flt$

# Applications

- Transaction Processing
- Electronic Data Interchange
- Programs that must interact but weren't designed together

- Airline reservation system
- Programmed Control Tower

## Features of Elephant

- I-O is in speech acts.
- Correctness involves proper performance of speech acts.
- Programs can refer to the past.
- Programs are represented as sentences of logic.
- *Accomplishment* and *input-output* specifications.

# ALGOL 48 AND ALGOL 50

Algol 60 program for multiplication by addition (The numbers in the left margin are not part of the program.)

```
0    start :  p := 0;
1             i := n;
2    loop :   if i = 0 then go to done;
3             p := p + m;
4             i := i − 1;
5             go to loop;
6    done :
```

Correctness condition: The program will eventually reach the label *done* at which point the variable $p$ will have the value $mn$. This is conventionally proved using the inductive assertion method, attaching the assertion $p = m(n - i)$ to the label *loop*. It is shown that this property is preserved in the loop. One proves that the program terminates by showing that the variable $i$ counts down from $n$ to $0$.

Algol 48 version of the Algol 60 program

$$p(t+1) = \textbf{if } pc(t) = 0 \textbf{ then } 0$$
$$\textbf{else if } pc(t) = 3 \textbf{ then } p(t) + m\,,$$
$$\textbf{else } p(t)$$

$$i(t+1) = \textbf{if } pc(t) = 1 \textbf{ then } n$$
$$\textbf{else if } pc(t) = 4 \textbf{ then } i(t) - 1\,,$$
$$\textbf{else } i(t)$$

and

$$pc(t+1) = \textbf{if } pc(t) = 2 \wedge i(t) = 0 \textbf{ then } 6$$
$$\textbf{else if } pc(t) = 5 \textbf{ then } 2$$
$$\textbf{else } pc(t) + 1$$

Its correctness is represented by the sentence

$$\forall n(n \geq 0 \supset \forall t(pc(t) = 0 \supset \exists t'(t' > t \wedge pc(t') = 6 \wedge p(t') = mn))).$$

This may be proved from the sentences representing the program supplemented by the axioms of arithmetic and the axiom schema of mathematical induction. Use mathematical induction on $n$ applied to a formula involving $p(t) = m(n - i(t))$.

Algol 50 version of the Algol 60 program

Preliminary axioms and definitions:

$$c(var, a(var, val, \xi)) = val,$$

$$var1 \neq var2 \supset c(var2, a(var1, val, \xi)) = c(var2, \xi),$$

$$a(var, val2, a(var, val1, \xi)) = a(var, val2, \xi),$$

and

$$var1 \neq var2 \supset$$
$$a(var2, val2, a(var1, val1, \xi)) = a(var1, val1, a(var2, val2, \xi)).$$

The following definitions

$$step(\xi) = a(pc, c(pc, \xi) + 1, \xi),$$

$$goto(label, \xi) = a(pc, label, \xi).$$

shorten the expression of programs. We mustn't forget to say $p \neq i \wedge p \neq pc \wedge i \neq pc$.

$$
\begin{aligned}
\forall t(\xi(t+1) = \ & \textbf{if } c(pc, \xi(t)) = \textit{start} \\
& \quad \textbf{then } \textit{step } a(p, 0, \xi(t)) \\
& \textbf{else if } c(pc, \xi(t)) = \textit{start} + 1 \\
& \quad \textbf{then } \textit{step } a(i, n, \xi(t)) \\
& \textbf{else if } c(pc, \xi(t)) = \textit{loop} \\
& \quad \textbf{then } (\textbf{if } c(i, \xi(t)) = 0 \textbf{ then } \textit{goto}(\textit{done}, \xi(t)) \\
& \quad \textbf{else } \textit{step } \xi(t)) \\
& \textbf{else if } c(pc, \xi(t)) = \textit{loop} + 1 \\
& \quad \textbf{then } \textit{step } a(p, c(p, \xi(t)) + m, \xi(t)) \\
& \textbf{else if } c(pc, \xi(t)) = \textit{loop} + 2 \\
& \quad \textbf{then } \textit{step } a(i, c(i, \xi(t)) - 1, \xi(t)) \\
& \textbf{else if } c(pc, \xi(t)) = \textit{loop} + 3 \\
& \quad \textbf{then } \textit{goto}(\textit{loop}, \xi(t)) \\
& \textbf{else } \xi(t+1))
\end{aligned}
$$

$$S = A + 5 \times S$$

ist(c,p)
value(c,exp)

Input-output and Accomplishment Specifications

- Illocutionary vs. perlocutionary speech acts

  I tell you the meeting is tomorrow.

  I inform you that the meeting is tomorrow.
  (You believe it.)

  I order you to come to the meeting.

  I get you to come to the meeting.

- Input-output and accomplishment program specifications.

  It says "Cleared to land" only when it perceives that the runway is clear.

  It says "Cleared to land" only when the runway is clear.

## DISCUSSION

**Rapporteur**: John E Dobson

### Lecture One

In the subsequent discussion, it was pointed out that Professor McCarthy had talked a lot about what should be in the curriculum but not much about how it should be done, and that style of presentation is as important as content. Professor McCarthy thought that the answer lay in the use of proper textbooks.

There was a lot of discussion on whether the demands of students would be better met by courses taught by computer scientists with a computer science bias or whether they should be taught by subject specialists. For example, physicists want to know about computing, by which they mean they wish to be taught Fortran. However, they sometimes get told by computer scientists, Fortran is no good, we will teach you something else. This is an example of computer scientists not appreciating how much physicists use Fortran. In this case, what is really required is somebody who understands both fields equally well. Again, it was pointed out that computer science students need to know a lot of mathematics so perhaps the only solution is to get mathematicians to teach them. Professor McCarthy observed that the dialogue between mathematicians and engineers oscillated, and that mathematicians tended to teach more than is required. He suggested leaving out number theory (except for cryptography) and geometry (except for robotics). He agreed that his proposed course contained a lot of material and so one needed to experiment to determine the correct level at which to pitch the topics. However, he did not agree that the course could be structured as a core body plus supplementary areas except with respect to advanced logic topics where it is probably not desirable in the near future to have computer science variants of these but to use professional courses.

There was agreement on the observation that a very good exercise is to get students to acquire a feel for the expressiveness of a language by getting them to use it to simulate something and that the course should include getting them to model something in logic.

On the topic of invariants in writing programs and in proving correctness of programs, Professor McCarthy commented that as someone with little familiarity with the topic, he hoped that not everyone had to do it. But some of his students needed proof theory.

### Lecture Two

In response to a question as to what needs to be given up in order to achieve the proposed language extensions, Professor McCarthy repeated that they probably could not be compiled since the computer would not be able to figure out appropriate data structures.

In response to a question about what happened to responsibilities, Professor McCarthy replied that responsibilities belong to the owner of the computer program. Obligations are delegated and it is desirable to prove that obligations are fulfilled in so far as they depend on the action of the program.

Professor McCarthy was unable to comment on the relation between his thoughts and the Co-ordinator of Terry Winograd and his colleagues or the BAN logic described by Roger Needham, but he could describe how the subject matter related to reality. If the axioms relating reality to the program's model of the world are wrong then problems will occur. But this is always true in other fields as well. You trust your life to other people's applied mathematics.