# ELEPHANT-2000—for the year 2005—well maybe 2015

John McCarthy, Stanford University
http://www-formal.stanford.edu/jmc/

March 10, 2008

I meant what I said,
And I said what I meant.
An elephant's faithful,
One hundred percent.

And also:

An elephant never forgets.

## NATURAL AND PROGRAMMING LANGUAGES

• Natural language has semantic procedural features absent in present programming languages. Some of them will be useful in programming languages if we can identify them.

• COBOL imitated a few surface syntactic features of English but made no semantic extension of (say) Fortran.

• A few hitherto unused semantic features of natural language are proposed for Elephant. Their use in programs moves work from the programmer to the compiler.

## PEOPLE AND ELEPHANT PROGRAMS REFER DIRECTLY TO THE PAST

• First Lead Example: A passenger has a reservation if he has made one and not cancelled it. The programmer writes nothing about an array or database for storing reservations. The compiler makes them. An Elephant source program never forgets.

• An Elephant compiler must include a designer of data structures adequate to implement the references to the past in a source program.

• Neither children nor generals can be bothered inventing data structures when they tell us what to do.

# ELEPHANT PROGRAMS INTERACT WITH OTHER PERSONS

• People use speech to act in their social and economic environments.

• These speech acts are not just assertions of facts.

• "I now pronounce you man and wife." "I hereby sentence you to be hanged by the neck until dead."

• Second lead example: "You have a reservation on flight UA 522 today at 7:35 pm." Such speech acts, if authorized, create obligations. An elephant is faithful, 100 percent.

# FEATURES OF ELEPHANT

• Communication inputs and outputs are in an I-O language whose sentences are meaningful speech acts identified in the language as promises, questions, answers, offers, acceptances, declinations, requests, permissions, etc.

• A promise will usually be represented by a string of symbols, but its meaning in the semantics of Elephant is as a promise, not just as a string. It creates an obligation to fulfill the promise, sometimes by the program itself.

• The correctness of programs is partly 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.

• 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 program that these obligations are exchanged as intended. These requirements also can be expressed by logical sentences.

• A full theory of the correctness of Elephant speech acts may require a theory of obligations, perhaps generalizing *deontic logic* to include predicates and quantifiers.

• Elephant programs themselves can be represented as sentences of logic. Their extensional properties follow from this representation without an intervening theory of programming or anything like Hoare axioms. We'll illustrate this for a simpler language than Elephant.

• Programs (not just in Elephant) 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.]

• Human speech acts involve intelligence. Elephant 2000 is on the borderline of AI, but we emphasize Elephant usages that do not require much AI.

## SAMPLE ELEPHANT FRAGMENTS

The notion of reservation:

$$\begin{aligned} &Has\text{-}Reservation(psgr, flt, t) \\ &\quad \equiv (\exists t' < t)(Makes\text{-}Reservation(psgr, flt, t') \\ &\qquad \wedge \neg(\exists t'')(t' < t'' < t \wedge Cancels\text{-}Reservation(psgr, flt, t''))) \end{aligned} \tag{1}$$

Handling the nested quantifiers typified by (1) may be difficult for compilers. There may be a better way of defining reservations without using data structures.

Letting the passenger on the airplane:

$$\begin{aligned} &\textbf{if } t = Time(flt) \\ &\quad \wedge\ Has\text{-}Reservation(psgr, flt, t) \\ &\quad \wedge \neg Physically\text{-}Full(flt, t) \\ &\qquad \textbf{then } Accept.Request(Admit(psgr, flt, t)) \end{aligned} \tag{2}$$

# SAMPLE COMMUNICATIONS FROM ELEPHANT PROGRAMS

- (Request (From: "John McCarthy") (Reservation: 2 (PM 7)))

- (Answer-to-request (From: "Denny's Palo Alto" (English: "We're out of business")))

XML would be only a little clumsier.

## REFERRING TO THE FUTURE?

• "The baggage handlers will be ready a half hour before the flight arrives." Not the same as "before the flight is scheduled to arrive."

• Referring to the future involves wishful thinking. Probably it should be left out of early versions of Elephant.

• Elephant doesn't prevent writing paradoxical programs, but a paradoxical program is a bad program.

# IMPLEMENTATION

• References to the past are implemented by suitable data structures.

• An interpreter may keep a journal of events and refer to it to compute references to the past.

• A compiler has to devise just the data structures the program needs to carry out the references to the past it actually makes. A program that can answer the question of whether a passenger has a reservation needs a database of current reservations. A program than can answer whether a passenger ever had a reservation needs a more elaborate database—again to be invented by the compiler.

# SPEECH ACTS

• Speech acts are sentences whose utterance has an effect apart from making an assertion that may be true or false. Their study was initiated by "ordinary language philosophers" who didn't like mathematical logic.

• "I now pronounce you man and wife. Congratulations". Contrariwise, "I sentence you to be hanged by the neck until dead."

• Speech acts include, offers, acceptances, statements, questions, promises, commands.

• One can also state, describe, assert, warn, complain, excuse, remark, comment, apologize, sentence, argue, persuade, propose marriage, threaten, challenge to a duel.

• On the input side there are *understand*, *realize*, *be offended by*. I haven't studied them much.

• It is worthwhile to design many computer outputs and inputs as speech acts and to so interpret statements by people and other computer programs.

## WHICH SPEECH ACTS DOES ELEPHANT NEED?

- Assertions. Assertions need to be truthful.

- Questions and answers to questions. Questions need to be comprehensible. Is that all? Answers need to be truthful and responsive.

- Requests of various strengths—from suggestions to commands.

- Offers, acceptances, and refusals.

Present interactive programs can often be regarded as performing speech acts—but in narrow pre-defined contexts, e.g. you are given a form with slots to fill in, preceded by a canned introduction.

## SPEECH ACTS HAVE SOCIAL, BUSINESS, AND LEGAL EFFECTS

- "I need a table for two at 7PM." has a business? effect.

- "You have a table for two at 7PM." creates an obligation.

- Programs that buy and sell goods and services make commitments and receive them.

- They undertake financial obligations on behalf of their owners.

- Their correct performance includes fulfilling their obligations and insisting that obligations to them be fulfilled. They are operating in society.

## INTERNAL SPEECH ACTS—COMMITMENTS

• A *commitment* is like a *promise* except that there is no output, i.e. it's an internal promise. Verifying a program includes verifying that it keeps its commitments.

• At some point during execution the program may execute a statement asserting the intention that from then on, the variable $x$ will remain less than the variable $y$.

• Is the execution correct so far? Will this subprocess terminate?

• Internal speech acts give rise to a form of *intrinsic correctness.*

• Should a program be made to hope or fear? Only for programs with considerable AI—at least with self-consciousness.

# TWO KINDS OF EXTERNAL SPECIFICATIONS

• The input-output specifications of a program depend on the program and the programming language. Their verification can be done using facts about these.

• The accomplishment specifications depend also on facts about the world, e.g. proving that an air traffic control program will prevent collisions also needs facts about airplanes and the behavior of pilots.

• Some arguments given against the utility of program verification have been based on confusing the two kinds.

• Speech act philosophers distinguish *illocutionary* and *perlocutionary* speech acts. Example: "He told her." vs. "He informed her." To some extent, these correspond to input-output and accomplishment specifications.

## DISCUSSION

• We avoid the philosophical problem "What is a promise?" by allowing a varieity of promise-like notions, according to utility.

• Promises suggested the notion of an internal commitment. For example, let $x$ and $y$ be variables in the program. At some point the program may execute

Commit: At some time in the future, $x > y$. This resembles a statement of temporal logic $F(x > y)$.

• Austin asked whether a promise is just a true report of a commitment and concluded *not*, because a promise creates an obligation.

# REFERENCES

"Towards a mathematical science of computation" by John McCarthy, IFIPS 1963, introduces abstract syntax.

*How to Do Things With Words* by J. L. Austin (Oxford 1962)

*Speech Acts* by John R. Searle, (Cambridge 1969)

This talk is based on *Elephant 2000: A Programming Language Based on Speech Acts* by John McCarthy. It is available as http://www-formal.stanford.edu/jmc/elephant.html.

Some pre-Elephant ideas are in
*Common Business Communication Language* by John McCarthy, http://www-formal.stanford.edu/jmc/cbcl.html.

in *Textverarbeitung und Burosysteme*, edited by Albert Endres and Jurgen Reetz, R. Oldenbourg Verlag, Munich and Vienna, 1982. The title of the book is in German, but all the papers are in English.