SOME LISP HISTORY AND SOME PROGRAM
LANGUAGE IDEAS

John McCarthy, Stanford University
Berlin, 2002 June 19

- I suppose I'm here because of Lisp, although I ha involved in the Lisp community for a very long time.

- Lisp is actively used, e.g. on the Deep Space 1 spac in the Orbitz airline reservation system, but I don't details.

- Fritz Kunze's Franz Inc. makes quite a good com called Allegro Common Lisp.

• Some important aspects of Lisp are not availabl
programming languages and systems. I don't know
used in the above applications.

• The original idea was to combine 1956 list processi
by Newell, Simon and Shaw with ideas from John Ba
tran.

• Herbert Gelernter at IBM undertook to implement M
sky's idea for a plane geometry theorem prover, and
list processing in Fortran. Gelernter and Carl Gerbe
oped FLPL.

• In 1958 Lisp was started at M.I.T. using recursion,
not feasible in Fortran. Lisp was intended for AI prog

• Lisp was intended to be compiled at first. Howeve
a universal Lisp function $eval$ in 1959 to show that
neater language for computability theory than Turing
Steve Russell pointed out that the universal functio
taken as an interpreter for pure Lisp, and hand-com
IBM 704 machine language.

DIFFERENTIATION—the motivating examp

The following example motivated

- recursion using conditional expressions
- lisp notation for algebraic expressions
- allowing functions as arguments with λ-expression s

diff(e, v) $\leftarrow$ if at e then
[if $e = v$ then 1 else 0]
else if car e $= PLUS$ then $PLUS$. $maplist(cdr\ e, \lambda u.dif$
else if car e $= TIMES$ then $PLUS$. $maplist(cdr\ e, \lambda u$
$maplist(cdr\ e, \lambda w.$ if u eq w then $diff(car\ u, v)$ else

ASPECTS OF LISP

- Lisp lists including lists of list are the appropriate repr
of symbolic expressions for computation—better thar
and better than XML.

- Lisp programs are Lisp data. Put abstractly, Lisp ha
for its own *abstract syntax.*

- Lisp programs, most conveniently pure Lisp funct
grams, are are described extensionally by first order s

- Many important properties of the functions can be
first order reasoning.

- Other important properties require *derived functior*

EXAMPLES OF LISP FUNCTIONAL PROGR

-

```
(defun append (u v)
(if
(null u)
v
(cons (car u) (append (cdr u) v))))
```

- $u \; * \; v \leftarrow \textbf{if } \textbf{n } u \textbf{ then } v \textbf{ else } \textbf{a } u \; . \; [\textbf{d } u \; * v]$ is a functional program.

- $(\forall u \; v)(u \; *' \; v = \textbf{if } \textbf{n } u \textbf{ then } v \textbf{ else } \textbf{a } u \; . \; [\textbf{d } u \; *$
equation for the function computed by the program.
correspondence is very convenient but sometimes cor

• The pure Lisp functional program as an equation pe
venient proofs in a first order theory that Lisp prog
their specifications. For example, it is easy to prove
duction that

$\forall u\ v.(u * v) * w = u * (v * w))$, i.e. that appending
associative operation.

LISP AND OTHER LANGUAGES

• Garbage collection, conditional expressions and rec
grams have been taken into other languages.

• LISP data structures have been imitated clumsily ir

(BUY item1 Item2 Item3)

<BUY> item1 item2 item3 </BUY>

• LISP programs having access to the abstract syn
program has not been imitated. This represents a lac
ination, but I admit I don't have convincing examples

DERIVED FUNCTIONS

The computational cost of a Lisp functional recursiv
is not determined by the extension of the function. W
$f91(x) \leftarrow$ if $x > 100$ then $x - 10$ else $f91(f91(x + 1$
and $ff91(x) \leftarrow$ if $x > 100$ then $x - 10$ else 91.

We have $\forall x. f91'(x) = ff91'(x))$, but clearly the func
grams are different computationally. Suppose we are
in how many times the $+$ operation is executed in
$f91(x)$. This is given by $f91p'(x)$, where

$f91p(x) \leftarrow$ if $x > 100$ then 0 else $1 + f91p(f91(x$
$f91p(x + 11)$.

ELEPHANT 2000: a programming language for the
www.formal.stanford.edu/jmc/elephant.htm

- An elephant never forgets. An Elephant progra
"A passenger has a reservation in a situation s if he h
reservation and not cancelled it. The Elephant progra
specify a data structure to remember reservations. Th
must provide the necessary data structures so that
passenger has a reservation can be determined.

$$Has(passenger, reservation, s) \equiv$$
$$(\exists s' < s)Occurs(Makes(passenger, reservation), s)$$
$$\wedge\neg(\exists s'')(s < s'' < s' \wedge Occurs(Cancel(Passenger, reserv$$

• An elephant is faithful one hundred percent. A r
is a promise to let the passenger on the airplane if h
when he shows up. One kind of Elephant output stat
promise, and correct Elephant programs fulfill their p

• The Elephant language includes program stateme
commitments generalizing Floyd assertions, because
refer to the future, A correct Elephant program fulfil
mitments.

$$(\forall s > Now)(Value(X, s) > Value(X, Now))$$

• Elephant i-o input output statments are speech acts,
tions, requests, acceptances of proposals, answers to
Answers to questions should be true and responsive.

ALGOL 48

If we introduce time explicitly as distinct from th counter, Algolic programs can be written as sets of Here's an Algol 60 program for computing the prod two natural numbers.

$$
\begin{array}{rcl}
start: & & \\
i & := & n; \\
p & := & 0; \\
loop: & & if\ i = 0\ then\ go\ to\ done; \\
i & := & i - 1; \\
goto & & loop; \\
done: & &
\end{array}
$$

Here's what mathematicians might have written in 19
programming languages existed.

$$
\begin{aligned}
pc(0) &= 0; \\
i(t+1) &= \text{if } pc(t) = \\
\text{else if } pc(t) = 4 \ \text{ then } i(t) - 1 \text{ else } i(t); & \\
p(t+1) &= \text{if } pc(t) = \\
\text{else if } pc(t) = 5 \text{ then } p(t) + m \text{ else } p(t) & \\
pc(t+1) &= \text{if } pc(t) = \\
\text{else if } pc(t) = 5 \text{ then } 2 \text{ else } pc(t) + 1; &
\end{aligned}
$$

The proof that $\exists t.(t \geq 0 \wedge pc(t) = 6 \wedge p(t) = mn)$ fo
the sentences expressing the program and the laws of
i.e. no theory of program correctness is needed. Ho
proof ideas are essentially the same as those used to
an algolic program terminates and that the output
correct relation to the inputs. Amir Pnueli and Nissi
had this idea before I did, but they mistakenly aband
temporal logic.