Refutation of lambda λ-calculus and LISP

© Copyright 2018 by Colin James III All rights reserved.

Abstract: We evaluate McCarthy's lambda λ-calculus in six equations. Five of the equations are *not* tautologous, and one equation is a *trivial* tautology. McCarthy's three-valued logic is not bivalent on which the LISP programming language is implemented, and hence also flawed.

We assume the method and apparatus of Meth8/VŁ4 with Tautology as the designated *proof* value, **F** as contradiction, N as truthity (non-contingency), and C as falsity (contingency). The 16-valued truth table is row-major and horizontal, or repeating fragments of 128-tables, sometimes with table counts, for more variables. (See ersatz-systems.com.)

LET: ~ Not, ¬ ; + Or, ∨, ∪; - Not Or; & And, ∧, ∩; \ Not And;
> Imply, greater than, →, ⊢; < Not Imply, less than, ∈;
= Equivalent, ≡, ⊨; @ Not Equivalent, ≠;
% possibility, for one or some, ∃, ◊, M; # necessity, for every or all, ∀, □, L;
~(y < x) (x ≤ y), (x ⊆y); (p=p) T as tautology; (p@p) **F** as contradiction.

From: de Vries, F-J. (2018). Many-valued logics inside λ-calculus: Church's rescue of Russell with Böhm trees. arxiv.org/pdf/1810.07667.pdf fdv1@le.ac.uk

LET: p, q, r, s: M; N.

3.1 Encoding Boolean logic in λ-calculus
In "the History of Lisp" ... John McCarthy mentions his "invention of the true conditional expression [if *M* then *N*1 else *N*2] which evaluates only one of *N1* and *N*2 according to whether *M* is true or false" and also his "desire for a programming language that would allow its use" in the period 1957-8. He also recalls "the conditional expression interpretation of Boolean connectives" as one of the characterising ideas of LISP. By this he means concretely the if-then-else construct (when applied to Boolean expressions only) in combination with the truth values T and F can be used as a basis for propositional logic ... with the following natural definitions:

$\neg M \equiv$ if M then F else T (3.1.1.1.1)

~p=(p>((p@p)+(p=p))) ; **T**T**F**T **T**T**F**T **T**T**F**T **T**T**F**T (3.1.1.1.2)

$M \wedge N \equiv$ if M then N else M (3.1.1.2.1)

(p&q)=((p>q)+p) ; **FFF**T **FFF**T **FFF**T **FFF**T (3.1.1.2.2)

$M \vee N \equiv$ if M then M else N (3.1.1.3.1)

(p+q)=((p>p)+q) ; **F**TTT **F**TTT **F**TTT **F**TTT (3.1.1.3.2)

$M \rightarrow N \equiv$ if M then N else T (3.1.1.4.1)

(p>q)=((p>q)+(p=p)) ; `TFTT TFTT TFTT TFTT` (3.1.1.4.2)

Remark 3.1: Eqs. 3.1.1.x.2 as rendered are not tautologous. This means lambda calculus as conceived and as implemented in the programming language LISP (LISt Processor) is not bivalent, and hence refuted.

... It is easy to see that if-then-else behaves as intended in this encoding. When B reduces to T and F, we have respectively:

if T then M else $N \rightarrow\rightarrow M$ (3.1.2.1.1)

(p=p)>(p+(q>p)) ; `TTFT TTFT TTFT TTFT` (3.1.2.1.2)

if F then M else $N \rightarrow\rightarrow N$ (3.1.2.2.1)

(p@p)>(p+(q>q)) ; `TTTT TTTT TTTT TTTT` (3.1.2.2.2)

Remark 3.1.2: Eq. 3.1.2.2.2 is a *trivial* tautology because (q>q) as the antecedent reduces to if False, then (True or True).

... The set of finite propositions can be defined formally with a[n] inductive syntax ... It is not hard to prove by induction that all closed finite propositions have a unique finite normal form:

Lemma 3.1. *Let φ be a finite closed proposition. Then φ has a unique finite normal form,which is either* T *or* F.

Remark 3.1.4: Lemma 3.1 does not help with or follow from Eqs. 3.1.1 or 3.1.2.

3.3 Encoding three-valued McCarthy logic with help of Böhm trees McCarthy's three-valued sequential three-valued propositional logic [from Fig. 2]: ¬T=F; ¬F=T; ¬⊥=⊥.

Remark 3.3: The values ¬⊥=⊥ are *not* tautologous, hence rendering this line of reasoning as not bivalent and rejected.

Because the five non-trivial Eqs. are *not* tautologous and the three-valued logic is non-bivalent, we reject lambda calculus and LISP on which many theorem provers are implemented.