Proofs

about programs, proofs as programs, programs as proofs!

Katherine Ye, !!con 2014

CORRECTNESS OF A COMPILER FOR ARITHMETIC EXPRESSIONS*

JOHN McCARTHY and JAMES PAINTER

1967

1 Introduction

This paper contains a proof of the correctness of a simple compiling algorithm for compiling arithmetic expressions into machine language.

The definition of correctness, the formalism used to express the description of source language, object language and compiler, and the methods of proof are all intended to serve as prototypes for the more complicated task of proving the correctness of usable compilers. The ultimate goal, as outlined in reference [1], [2], [3] and [4] is to make it possible to use a computer to check proofs that compilers are correct.

The concepts of abstract syntax, state vector, the use of an interpreter for defining the semantics of a programming language, and the definition of correctness of a compiler are all the same as in [3]. The present paper, however, is the first in which the correctness of a compiler is proved.

The expressions dealt with in this paper are formed from constants and variables. The only operation allowed is a binary + although no change in method would be required to include any other binary operations. An example of an expression that can be compiled is

$$(x + 3) + (x + (y + 2))$$

*This is a reprint with minor changes of "Correctness of a Compiler for Arithmetic Expressions" by John McCarthy and James Painter which was published in MATHEMATICAL ASPECTS OF COMPUTER SCIENCE 1, which was Volume 19 of *Proceedings of Symposia in Applied Mathematics* and published by the American Mathematical Society in 1967

```
(* ###########################################################
(** ** Type Preservation *)

(** The second critical property of typing is that, when a well-type
    term takes a step, the result is also a well-typed term.

    This theorem is often called the _subject reduction_ property,
    because it tells us what happens when the "subject" of the typin
    relation is reduced.  This terminology comes from thinking of
    typing statements as sentences, where the term is the subject an
    the type is the predicate. *)

Theorem preservation : forall t t' T,
  |- t \in T ->
  t ==> t' ->
  |- t' \in T.

(** **** Exercise: 2 stars (finish_preservation) *)
(** Complete the formal proof of the [preservation] property.  (Agai
    make sure you understand the informal proof fragment in the
    following exercise first.) *)

Proof with auto.
  intros t t' T HT HE.
  generalize dependent t'.
  has_type_cases (induction HT) Case;
        (* every case needs to introduce a couple of things *)
        intros t' HE;
        (* and we can deal with several impossible
           cases all at once *)
        try (solve by inversion).
    Case "T_If". inversion HE; subst.
      SCase "ST_IFTrue". assumption.
      SCase "ST_IfFalse". assumption.
      SCase "ST_If". apply T_If; try assumption.
        apply IHHT1; assumption.
    (* FILL IN HERE *) Admitted.
(** □ *)

(** **** Exercise: 3 stars, advanced (finish_preservation_informal)
(** Complete the following proof: *)

(** _Theorem_: If [|- t \in T] and [t ==> t'], then [|- t' \in T]. *
```

"Software Foundations" on right

From a paper proof of correctness to a computer-checked proof of correctness of our optimizations. (I highly recommend checking out "Software Foundations." It's an interactive textbook.)

You're a compiler.

Let x, c, end1, end2 be natural numbers.

```
if !(x < c)
  then end1
  else end2
```

Let x, c, end1, end2 be natural numbers.

Optimize?

```
if !(x < c)
  then end1
  else end2
```

Can you remove the negation and switch the two possible return results?

Let x, c, end1, end2 be natural numbers.
Then for all x, c, end1, end2, is this true?

?=

```
if !(x < c)
  then end1
  else end2
```

```
if (x < c)
  then end2
  else end1
```

Proof: 2 possibilities

Are we right? Prove that they are equal.

(x < c) == ?

---

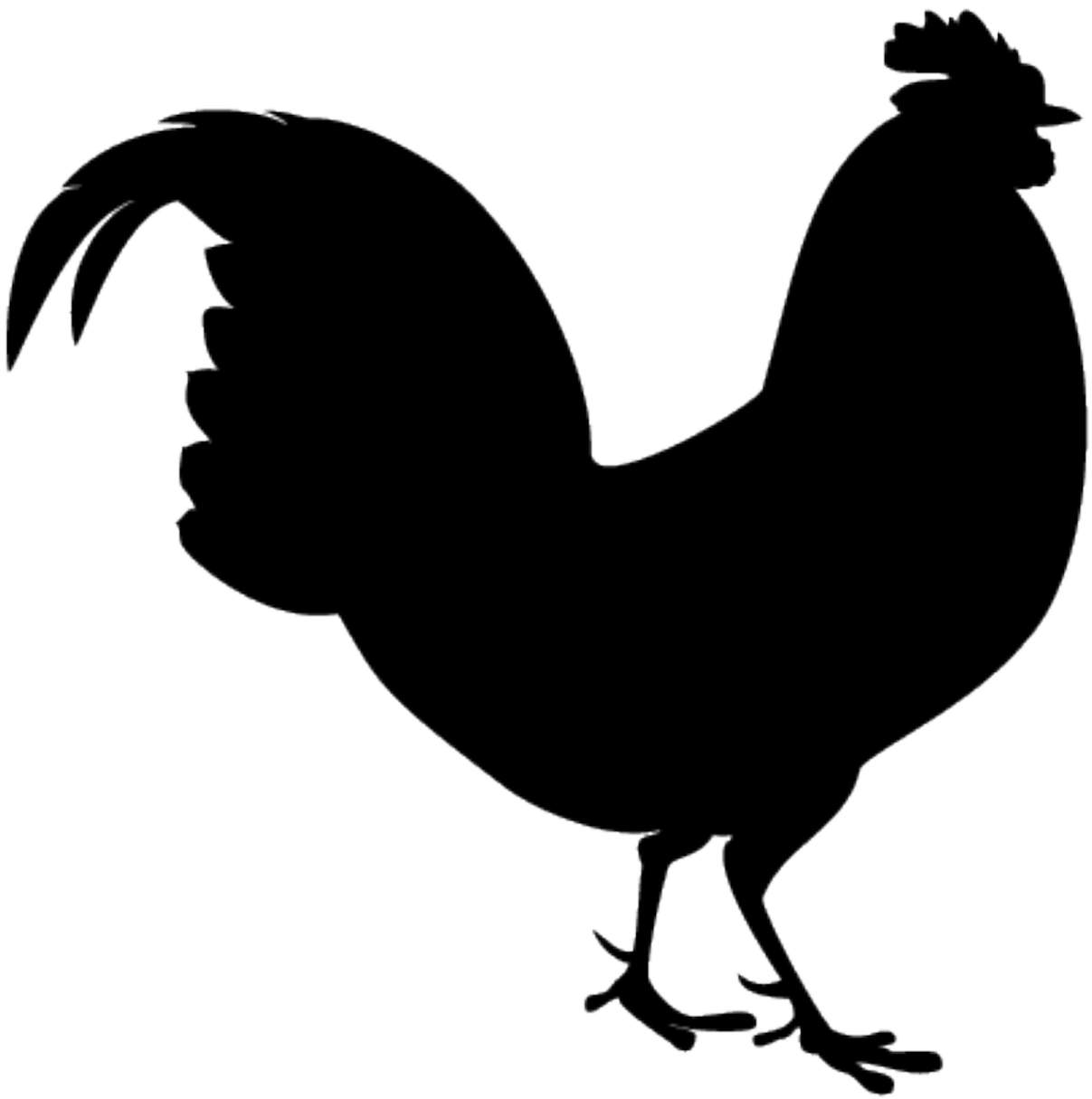
1. `if !(x < c)`
   `then end1`
   `else end2`

2. `if (x < c)`
   `then end2`
   `else end1`

value of (x < c)

Code used		TRUE	FALSE	Code output
	#1	end2	end1	
	#2	end2	end1	

Code equality!
(compiler optimizations)



Coq: interactive theorem proving!

We've convinced our human-selves with the proof, but not our compiler-selves. Why not check the proof with a computer?

“Coq: the world’s best computer game”

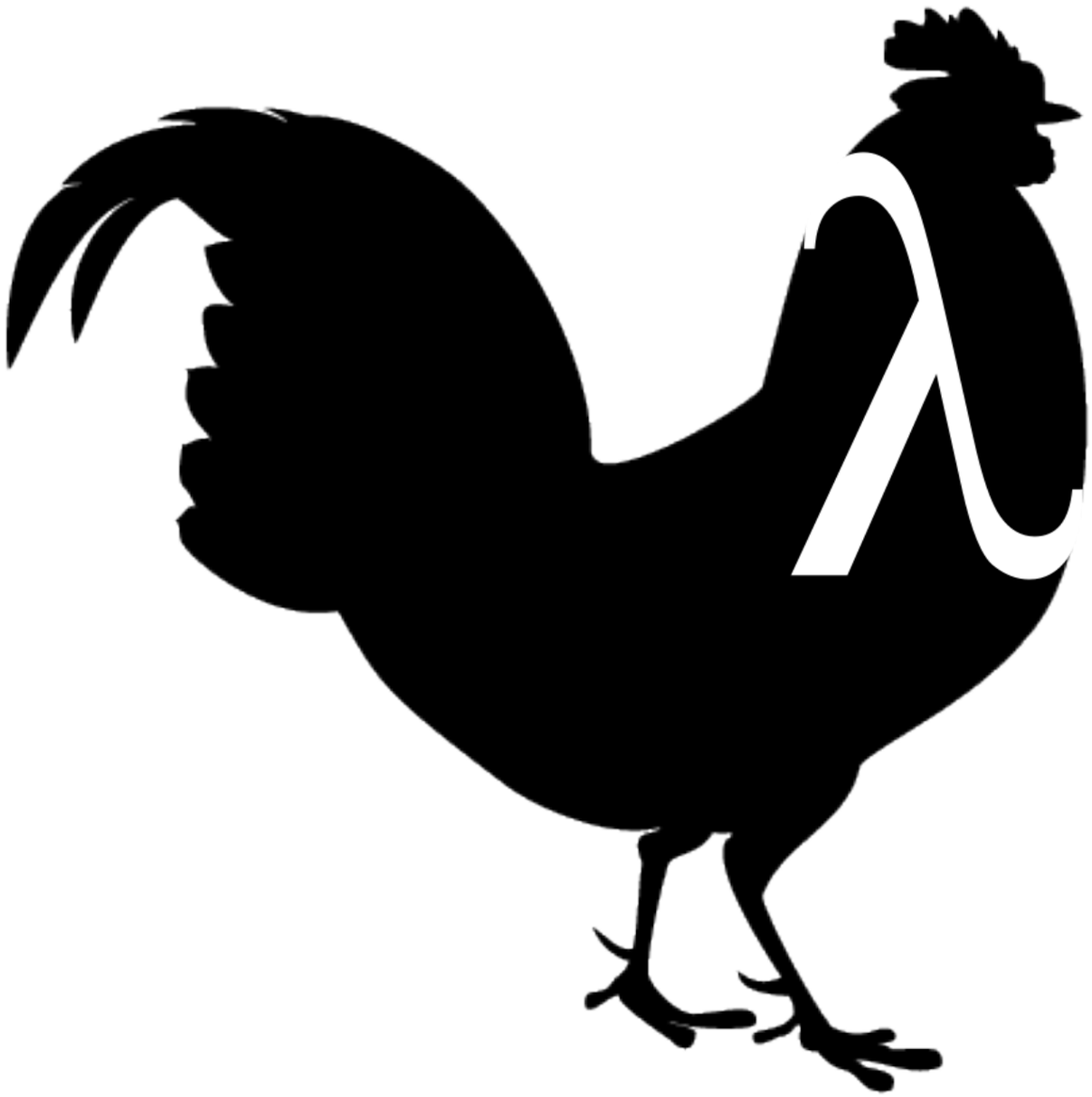
http://golem.ph.utexas.edu/category/2012/06/the_gamification_of_higher_cat.html

Live-coding interlude

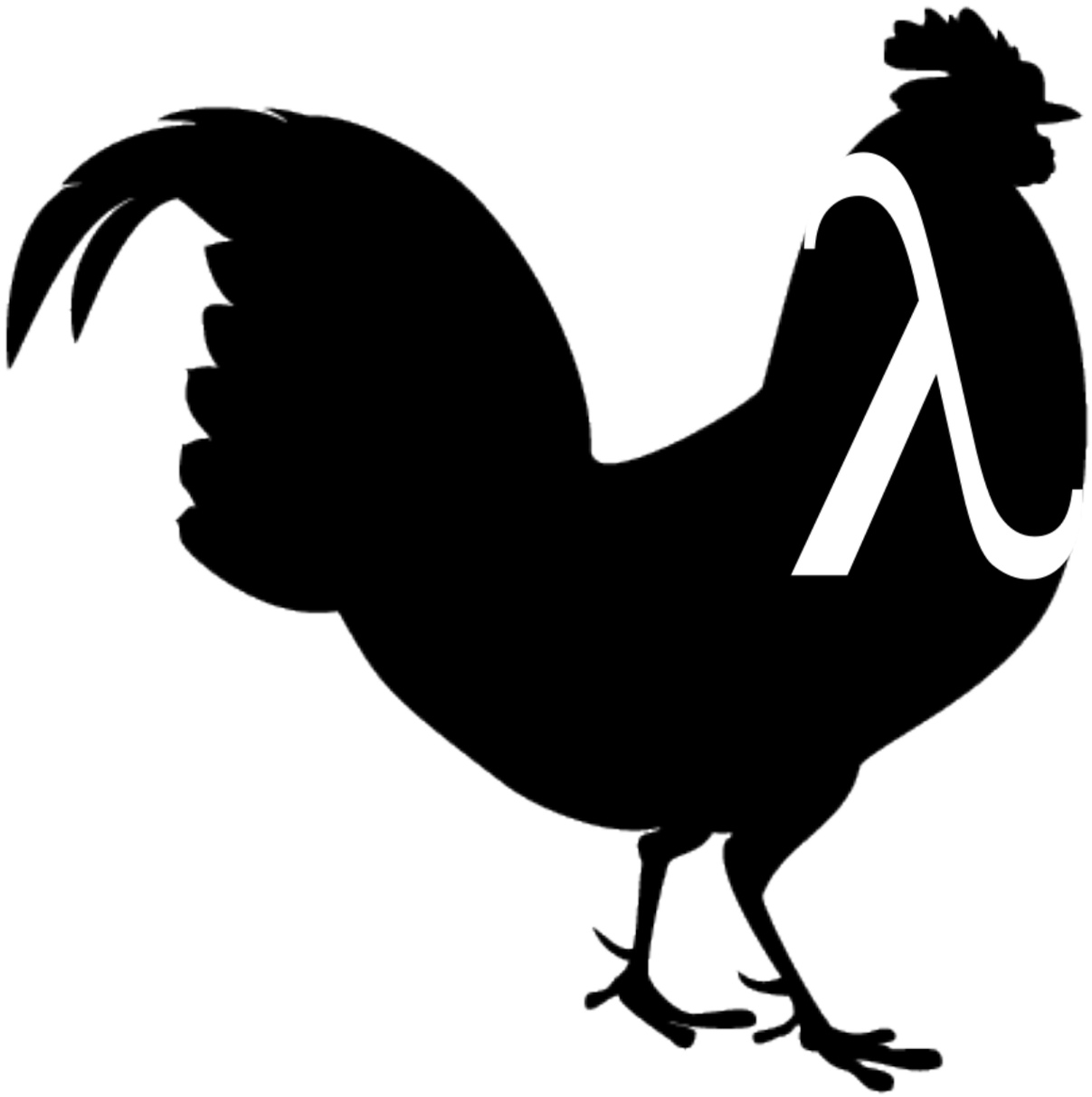
(see katherineye.com for the code)

Qed.

How does Coq work?



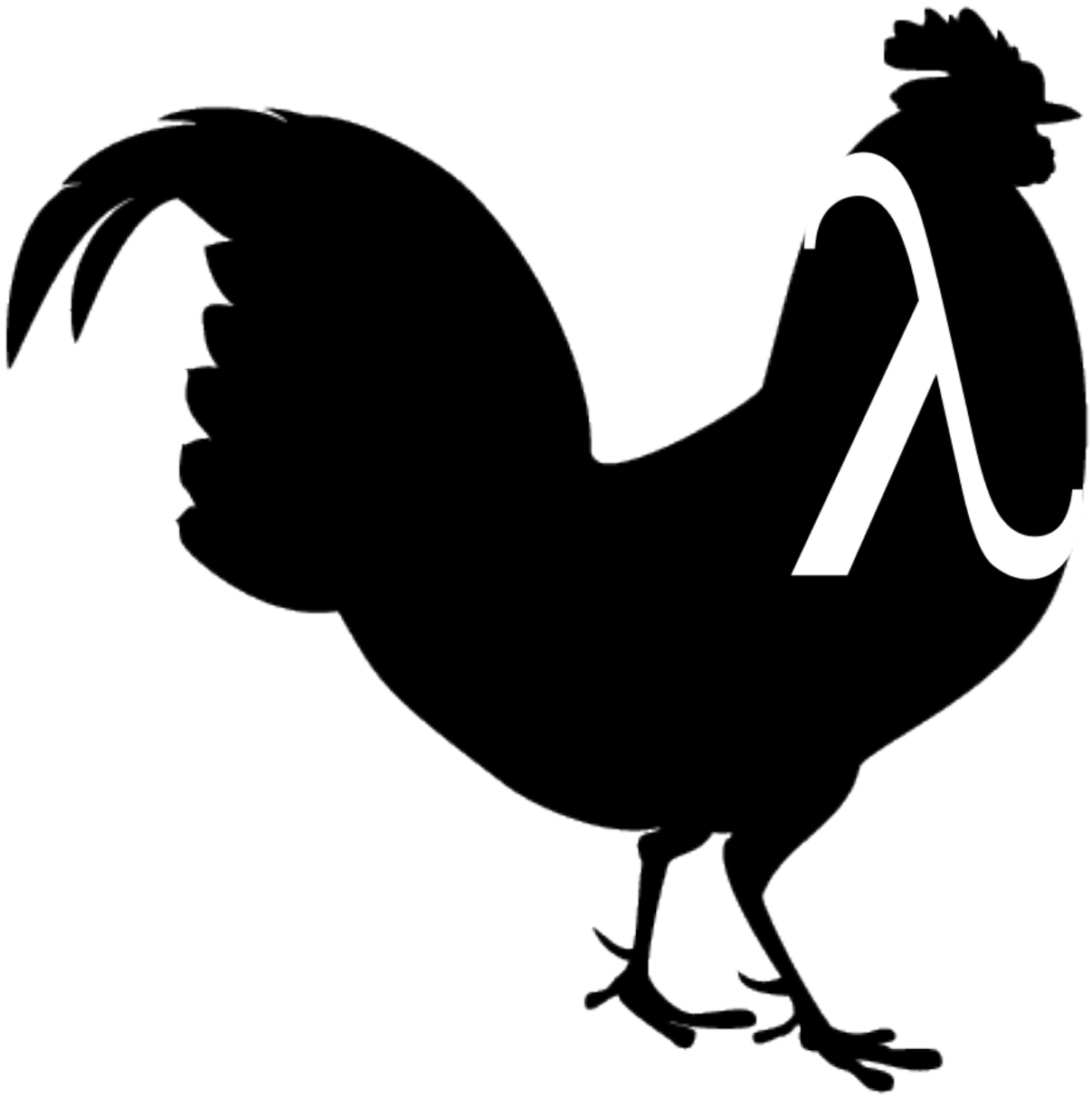
Coq = CoC



Coq = CoC

= Calculus of Constructions

The CoC is a generalization of a correspondence that we'll talk about now.



Math profs HATE it:

The one weird trick for interpreting your proofs as programs (and vice versa)

A type <-> a proposition

<->: corresponds to

a -> a

a -> a

- Given anything of type a, I can give you something of type a
- identity
- f x = x

Interpreted as the type of a function

a -> a

- Given anything of type a, I can give you something of type a
- identity
- f x = x

- Assuming hypothesis a, I can show that a is true
- Trivial: a was assumed to be true!

Interpreted as a proposition for which a proof may or may not exist

a -> a

- Given anything of type a, I can give you something of type a
- identity
- f x = x

- Assuming hypothesis a, I can show that a is true
- Trivial: a was assumed to be true!

Isomorphism

Program <-> proof

(“value”)

Examine our proof

Fancy name: Curry-Howard isomorphism

You can see that our proof, written in Coq's tactic language, is internally represented as something that looks very much like a function. It's written in Gallina, Coq's internal language.

Lots of active research!

- verify compiler (CompCert)
- verify compiler optimizations (Vellvm, of LLVM)

- verify cryptographic code (SHA-256 in OpenSSL, Appel)
- create new foundations of mathematics: homotopy type theory

- writing the game 2048 in Coq (Laurent Théry)

I'd like to assert that the last application is the most important. (code not by me, but can be seen on my site: katherineye.com)

Thanks!

Katherine Ye
@hypotext