

AlgolML

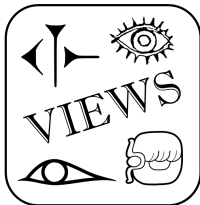
An Executable Notation for Algorithms and Programs

Kevin Graaf

Visual Interactions in Early Writing Systems (VIEWS),
University of Cambridge*

*Starting 1 October 2026

13 August 2026



100% Human-made

Everything in this presentation is 100% human-made.

No “AI.”

No LLMs.

Jane Q. Digital-Humanist wants to encode some document or artifact with “computational” elements. This could be:

- A game with some kind of “logic”
- Interactive fiction (e.g. Zork)
- Hypermedia
- A process
- Printed code

In any case, helpful to have standard formalism to computationally “gloss” the meaning of such components

If you find these questions interesting, join the Computable Media SIG!

What are our options?

She could:

- Use a standard or common PL: Python, Javascript, etc.
- Use pseudocode
- Write a prose description

None of these options are good!

Common PLs Are Not Archival Quality

- All common PLs are huge, complex, and rely on a transitive closure of *millions* of lines of code (between language itself, packages, OS)
- **Zero guarantee any code written in these will be runnable in 5, 10, 50, 100 years**
- Moreover, zero isolation of computational *environment* (in PL theory we call this a coeffect)
- Personally, I've got maybe a 50% success rate installing Python package, tops
- Shout to *Minimal Computing* movement, exactly the right intuitions!

The “Pseudo” in Pseudocode

- Pseudocode is not code, has no formal (or even informal) semantics
- Consequently, not executable, no particular meaning
- It is **far** too easy for humans to make “obvious” logical jumps
- This is why formalization is hard!
- Prose descriptions are inadequate, for the same reasons
- At the same time, notation shouldn't be too low level, i.e. pedantic

A Brief Introduction to Algol

Here is a language so far ahead of its time, that it was not only an improvement on its predecessors, but also on on nearly all its successors.
— Tony Hoare, “Hints on Programming Language Design”

- CSists early in this discipline thought about this problem too!
- Result of this work was the “International Algorithmic Language” (IAL) or Algol 58
- Algol 58 was revised and clarified in the form of Algol 60
- The most important language you’ve never heard of
- So many things we take for granted: structured control flow, lexical scope, and more—pioneered by Algol
- Nearly all widely used languages owe something to Algol 60
- Used as the systems language of the best computer ever designed, the Burroughs Large Systems, still in use today!
- Not without warts—Call-By-Name Evaluation (think Haskell) yet impure, many parts left unstandardized

```

procedure euler (fct, sum, eps, tim) ; value eps, tim ;
integer tim ; real procedure fct ; real sum, eps ;
comment euler computes the sum of fct(i) for i from zero up to
infinity by means of a suitably refined euler transformation. The
summation is stopped as soon as tim times in succession the absolute
value of the terms of the transformed series are found to be
less than eps. Hence, one should provide a function fct with one
integer argument, an upper bound eps, and an integer tim. The
output is the sum sum. euler is particularly efficient in the case
of a slowly convergent or divergent alternating series. ;
begin integer i, k, n, t ; array m[0:15] ; real mn, mp, ds ;
i := n := t := 0 ; m[0] := fct(0) ; sum := m[0]/2 ;
nextterm: i := i+1 ; mn := fct(i) ;
    for k := 0 step 1 until n do
        begin mp := (mn+m[k])/2 ; m[k] := mn ;
            mn := mp end means ;
    if (abs(mn)<abs(m[n]))^(n<15) then
        begin ds := mn/2 ; n := n+1 ; m[n] :=
            mn end accept
    else ds := mn ;
    sum := sum + ds ;
    if abs(ds)<eps then t := t+1 else t := 0 ;
    if t<tim then go to nextterm
end euler

```

What about Algol 68?

See the signs of suspicion! Count the hemisemidemicolons! Screamer caps and invented gommars, quoites puntlost, forced to farce!

— *Finnegans Wake*, Book II, Episode 3

- Algol 60 had a successor!
- Brilliant, yet maddeningly complex
- Formalized in a unique but inscrutable manner
- At least one technical term borrowed from Joyce
- Currently undergoing a strange, fascinating renaissance
- Probably not the right choice here, but worthy of study

What Does Digital Preservation Means?

- Just as merely placing a text online is not archiving, so too with code
- In fact, the situation is much worse!
- Knowing English is enough to make use of English text (on some level)
- Knowing Python is *not* enough to run Python
- It is hard to overstate just how precarious and contingent our “modern” computing environment is
- The *only* way to ensure code is executable “forever” is to write it in a language so simple, anyone can implement that language
- Permacomputing community has tackled similar questions, worth exploring for those interested
- I want to promote the concept of “archival quality code” (by analogy to paper!)

- XML-based concrete syntax, leverage Invisible XML for more ergonomic alternatives
- But at the same time, allow for *rendering* as beautiful, 2D syntax
- Avoid the PL theorist desire for anything semantically *too* exotic
- Similarly, avoid static typing for maximal generality
- Allow for (subset of) MathML in expressions
- Play nicely with TEI, especially w.r.t. critical apparatus to document *changes* in code semantics

AlgolML Programming Model

- Dynamically typed, functional programming language similar to Lisp
- XML used as a universal data structure, analogous to the list in Lisp, string in Tcl, array in APL, etc.
- Simple—can be implemented in a few thousands lines of code at most
- Locally impure, globally pure—mutable state is possible, but AlgolML programs present as “pure” mathematical functions at interface level
- Exploring rationalized fexprs, a forgotten paradigm from early Lisp, as a simple, universal mechanism, drawing on work of John Shutt
- Fexprs are a bit like “run time macros,” account for most of the weirdness of R

A Taste of AlgolML

```
<algolml module="fibonacci">
  <function n="fib" recursive="tail">
    <arguments>
      <argument n="n"/>
      <argument n="p0"/>
      <argument n="p1"/>
    </arguments>
    <code>
      <switch><var n="n"/>
        <cond>
          <lt><var n="n"/><num v="0"/></lt>
          <num v="-1"/>
        </cond>
        <cond>
          <or>
            <eq><var n="n"/><num v="1"/></eq>
            <eq><var n="n"/><num v="0"/></eq>
          </or>
          <var n="p1"/>
        </cond>
        <else>
          <call>
            <var n="fib"/>
            <sub><var n="n"/><num v="1"/></sub>
            <add><var n="p1"/><var n="p0"/></add>
            <var n="p1"/>
          </call>
        </else>
      </switch>
    </code>
  </function>
```

```
<function name="main">
  <arguments>
    <argument name="n"/>
  </arguments>
  <code>
    <call>
      <var n="fib"/>
      <var n="n"/>
      <num n="0"/>
      <num n="0"/>
    </call>
  </code>
</function>
</algolml>
```

What's Next?

- AlgolML is *very* much a work in progress
- I want to hear from *you* about what kind of computable media you want to encode and what you need to do it
- I am currently writing an interpreter in Standard ML
- I'd love to see alternative implementations!
- Join the SIG!

Questions?

email: `kevingraaf@kbg.sh`

website: `https://kbg.sh`

code: `https://fossil.kbg.sh`