Programming languages based on formal structures

Dale Miller

Inria Saclay & LIX, École Polytechnique
Palaiseau, France

Autumn workshop II: Programming Languages and Notations
2 October 2019

Programs should have several properties

We increasingly demand that programs can:

- be efficiently executed (via compilers, interpreters, etc),
- be readable and maintainable,
- support the determination of time and space complexity
- be proved partially correctness,
  - decidable static analysis, e.g., types, abstract interpretation
  - support testing and model checking to help locate bugs
- be securely executed in adversarial environments, and
- be proved functional correctness (satisfies a specification).

Since we are speaking of properties of code, logic has roles to play.

Roles of logic in computing

- Computation-as-model: Computation happens, registers change, lights flash, etc. Logic is used to make statements about the dynamics.
  - classical/intuitionistic logic/arithmetic are typically used
  - Also, various modal/temporal logics, Hoare triples
  - Automated tools: model checkers and testing
- Computation-as-deduction: Computation is based on bits of logic: formulas, terms, proofs.
  - proof normalization (functional programming, Curry-Howard isomorphism)
  - proof search (logic programming, relational programming)

Examples of functional programs

Example languages: LISP, Scheme, ML, Haskell, OCaml, ...

```
(add (mult 34 56) 96) : int
```

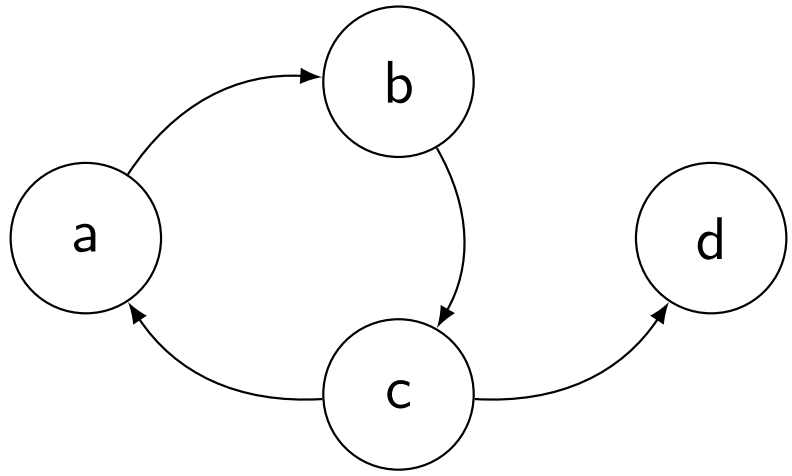
```
(foldr (lambda x y (x + y)) 0
       (map sin [0.1, 0.3, 1.2])) : real
   --> (sin 0.1) + (sin 0.3) + (sin 1.2)
```

```
foldr : (A -> B -> B) -> B -> A list -> B
```

Code is a proof of its type.

Computation is rewriting to normal form.

An example of a logic program




```
adj(a,b).    adj(b,c).    adj(c,a).    adj(c,d).
path(X,Y) :- adj(X,Y).
path(X,Z) :- adj(X,Y), path(Y,Z).
```

$$\dfrac{\dfrac{}{\mathcal{P} \vdash adj(a,b)} \quad \dfrac{\dfrac{}{\mathcal{P} \vdash adj(b,c)} \quad \dfrac{\dfrac{}{\mathcal{P} \vdash adj(c,d)}}{\mathcal{P} \vdash path(c,d)}}{\mathcal{P} \vdash path(b,d)}}{\mathcal{P} \vdash path(a,d)}$$

What is a logic?

Examples: first-order and higher-order versions of classical and intuitionistic logics.

Logics have rich meta-theory

- soundness and completeness for some collection of models (proof and truth both describe the set of theorems)
- natural deduction, sequent calculus
- proofs are PTIME checkable
- algorithms exist for proof search (unification, resolution, etc).

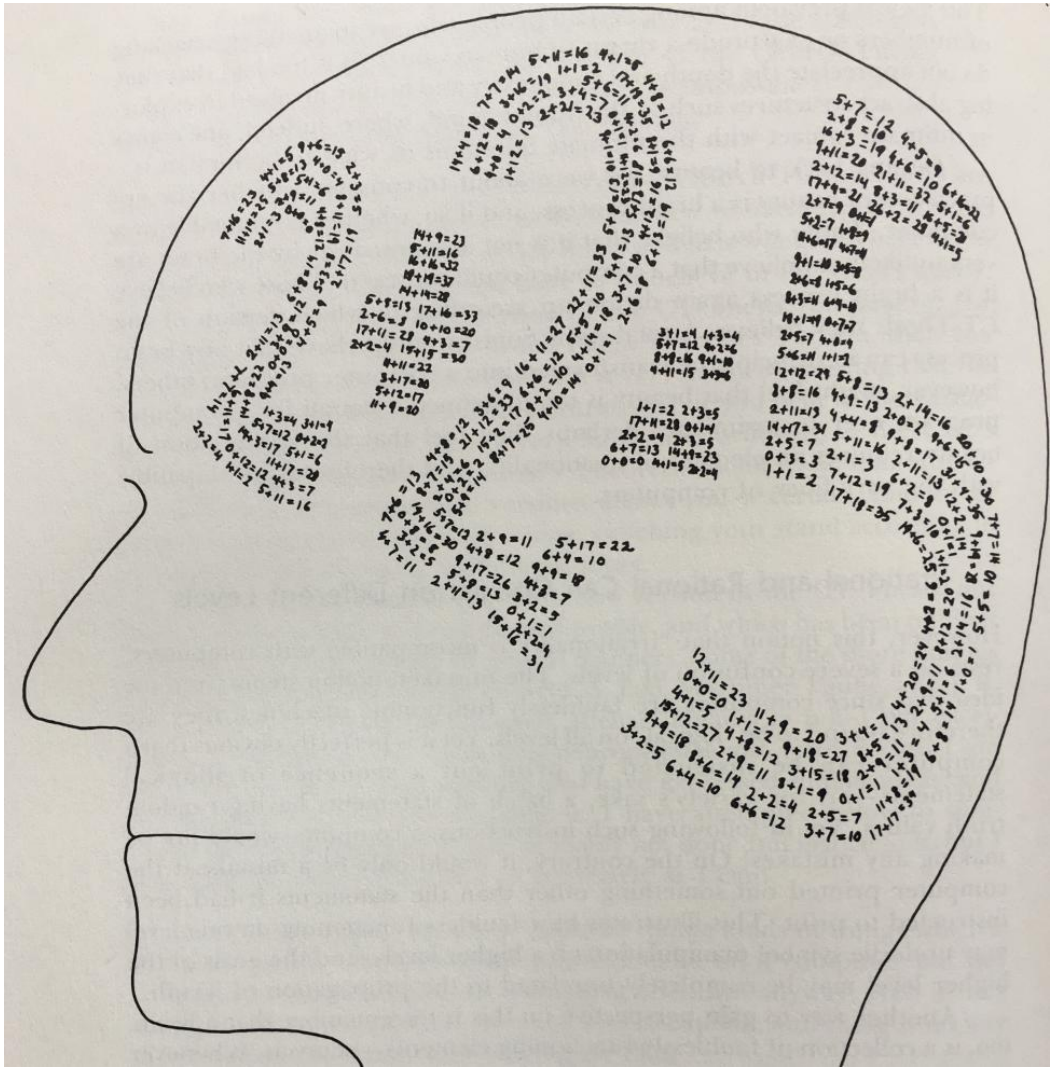
At a minimum, a logic should be consistent. Often, we require a cut-elimination theorem.

Is linear logic a logic?

- No good model theory semantics for full linear logic exists.
- Linear logic has a cut-elimination theorem.
- We return to this question later.

Naive reason for basing a programming language on logic

Using programs made of logic must yield correct programs.



Refutation by picture

From Hofstadter's
Gödel, Escher, Bach

A wishful perspective

- Kowalski (1979):

$$Algorithm = Logic + Control$$

- The Prolog community in the 80s/90s:

$$Program = Logic + Control + modules + I/O + Meta + \ldots$$

  The weak logic of Horn clauses was overwhelmed with not-so-logical stuff.

- What about

$$Program = Logic \quad ?$$

  We return to this question later.

Control issues are also familiar in the functional programming setting: call-by-value, call-by-name, call-by need, etc.

Benefits of the computation-as-deduction paradigms

Logic provides new perspectives on computing: "logic variable", multiset rewriting, non-determinism, binder mobility, ...

Logic has traditionally not concerned itself with resources (memory allocation, threads, scheduling, etc) so new designs in software were forced.

For example, when references to memory resources are removed from the language, one uses a garbage collector. If your GC is correct, then **any** program using that collector will not have memory leakage, buffer overflows, etc.

Part of the success of Java is its use of a garbage collector. Mismanaging memory was the first on the *2019 CWE Top 25 Most Dangerous Software Errors*.

Logics have various deep properties

Logics generally have a well-developed literature.

- First-order/higher-order and classical/intuitionistic logics are permanent standards. Fifty years from now, anyone can, in principle, understand the details of that standard. The literature will probably improve by then also.
- A collection of rich results exist about various logics: proof systems, model theories, soundness/completeness, cut-elimination, Herbrand's theorem, etc.
- Multiple ways to implement logics have been described: unification, skolemization, resolution refutations, etc.

If only programming languages were so well developed!

These benefits can come also from other systems

Logic is not alone in providing these benefits. Consider:

- lambda-calculus
- finite state machines, regular languages (tokenizers, search queries)
- context-free languages, push down automata (parsers, attribute grammars)
- CSP (Hoare); Occam
- CCS, π-calculus by Milner (specification of concurrency)
- Spi-calculus: security protocols
- Relational calculus, datalog (database programming)
- Petri nets, multiset rewriting

None of these are considered logics although their relationships to various logics have been studied.

Robin Milner (1934-2010) is an interesting example.

- In CCS and the π-calculus, bisimulation is defined co-inductively and is taken as an observational equivalent. He proved that bisimulation is a congruence. Hence, algebraic style reasoning can be applied to the π-calculus.
- Bigraphs made reference to elementary category theory to help organize the definitions and results.
- The meaning of many of his systems used Plotkin's structural operational semantics, a kind of logic programming paradigm.

An example: Computing with multisets

Multisets are sets where elements have multiplicities.
Equivalently, lists where the order of elements in the list is ignored.

- $\{a, a, b\} \longrightarrow \{b\}$
- $\{p1, lock\} \longrightarrow \{q1, lock\}$, $\{p2, lock\} \longrightarrow \{q2, lock\}$
  There are two sequential ways to rewrite

$$\{p1, p2, lock\} \longrightarrow^* \{q1, q2, lock\}$$

- $\{a(x), a(y)\} \longrightarrow \{a(x)\}$ if $x \geq y$. In general,
  $\{a(x_1), \ldots, a(x_n)\} \longrightarrow^* \{a(x_0)\}$ iff $x_0 = \max(x_1, \ldots, x_n)$.

Foundation of various computing paradigms: Petri nets, Linda coordination language, security protocols, etc.

Is this logic?

What is a formal structure? First try

A mathematically described structure useful for computing.

Obviously, not all such formal structures are useful for designing programming languages.

Turing machines are formally defined objects. They have played an important role in computer science for at least two reasons.

- They obviously capture a notion of computation, which is a prerequisite for establishing a negative result (the halting problem).
- Turing machines have been useful for providing formal definitions of the time and space hierarchies.

Turing machines are not a good starting place for programming languages. They are too low-level, they are not modular, they admit almost no abstractions, etc.

What is a formal structure? Second try

Mathematically defined structures involving syntax and which allow for rich manipulation of and reasoning with such syntax.

- Classical and intuitionistic logics are formal structures in this sense.
  - First-order logic: model theory (truth) = theorems (proofs).
  - Provability often has multiple equivalent characterizations using resolution, natural deduction, sequent calculus, etc.
  - Meaningfully manipulated of syntax: logical equivalence, De Morgan duality, skolemization, conjunctive normal form, etc.
- There are a number of other kinds of formal systems.
  - Böhm separation theorem and Church-Rosser confluence results for the λ-calculus.
  - Bisimulation is a congruence (various concurrent formalisms).
  - Game theory and category theory contain subtopics that are such formal structures.

Is Linear Logic a logic?

It has a good proof theory but no matching model theoretic semantics. It is clearly a formal system with many deep properties and applications.

- Provability and cut-free provability describe the same provable formulas.
- LL provides various ways to encode intuitionistic and classical logic proofs.
- LL has been used to encode Petri nets, multiset rewriting, and (subsets of) the π-calculus.

"Is it a Logic?" seems to be debatable.
"Is it a formal system useful for computing?" Certainly.

Trade-offs when using formal structures

Being a general-purpose programming language based on a formal system usually results in some ad hoc (breaking with the formal system) features being inserted. Recall:

$$Algorithm = Logic + Control$$

$$Program = Logic + Control + modules + I/O + Meta + \ldots$$

To the extent that large parts of code remains "pure", i.e.,

$$Program = Logic$$

is the extent to which

- compiler optimizations can be done,
- static analysis is effective, and
- formal proofs are supported.

Evidence that logic is missing the point

Maybe the design of logic goes a bit too far.

- Assume that the state of a computation is encoded as a formula. When computation is modelled as state S_1 evolving to S_2, which implication should we select? $S_1 \vdash S_2$ or $S_2 \vdash S_1$. I.e., are we progressing to truth or to false? This choice is arbitrary.
- Intuitionistic logic forces an asymmetry: multiple inputs but only one output.

Other approaches where these choices are not imposed.

- Game theory might provide a more neutral approach. Depending on who has a wining strategy, you have proved or refuted a formula.
- Linear logic allows for multiple inputs, multiple outputs.

More evidence that logic is missing the point

Assume that a process (in concurrency theory) is encoded as a formula.

The fact that process P_1 is simulated by process P_2 can sometimes be captured by $P_1 \vdash P_2$.

Thus, if P_2 is also simulated by process P_1, then it must be the case that P_1 and P_2 are logically equivalence.

But it is well-known that such mutual simulation is weaker than bisimulation (a central equivalence in process calculi).

Thus, bisimulation (in the general setting) cannot be encoded using formulas and logical entailment (without some new breakthroughs).

A future for logic and programming languages

The story between logic and programming has mostly been one of making logic more expressive in order to capture more computational aspects.

$$\textit{classical} \longrightarrow \textit{intuitionistic} \longrightarrow \textit{linear}$$

There does seem to be a push in the opposite direction.
More and more principle of logic seem to be incorporated into programming languages.

- strong static properties (e.g, typing and abstract interpretation)
- richer abstractions (abstract datatypes, generic functions)
- equality reasoning via congruences (e.g., bisimulations).

In such a case, programming languages are moving closer to what one might call logic (but not the conventional logics!).

Conclusions

Modern demands on programming languages and programs has changed a great deal in recent decades. Formal properties enable solving many of these demands. (Correctness, security, compilation targeting multiple architectures, testing, optimizations, etc)

The "computation-as-deduction" approach provides exciting but limited interchange between logic and computation. That interchange is likely to expand but probably not a lot.

The change from "logic" to "formal systems" more generally seems forced.

Some programming languages are likely to evolve so that they embrace more and more formal properties.