Finding Unity in Computational Logic

Dale Miller
Parsifal team
INRIA - Saclay & LIX, Ecole Polytechnique
Palaiseau, France

Software and hardware correctness is importance

There is an old chestnut for justifying strong foundations.

> You can't build a tall building on a sandy beach.

The modern update requires moving off the beach.

> If you are in a canoe, a small leak might be okay; if you are in a submarine, a small leak is lethal.

Plugging your computer into the internet is similar to descending into the depth of the sea: if there is a crack in your security, it will be exploited quickly.

One cannot be relaxed about buggy software and hardware anymore....

Logic as a framework for correctness

We expect logic to be *secure* and *universal*.

Leibniz hoped for a universal formal language in which all disputes could be answered simply by

> Let us calculate ... and see who is right.

Logic is secure (sound) and universal (complete): e.g., first-order, classical logic.

On such a secure foundation, one can build set theory and much of mathematics.

A brief history of computational logic

An early dream for automated logic:

> Let us implement logic (and why not, since its syntactic side is organized around a small set of proof principles): we shall then have an implementation of mathematics.

This earliest work yielded many important lessons:

- Roles of unification, backtracking search, etc.
- Only the simplest problems could be captured in practice with universal methods.

Revising the goals of pure automation

Model checking and *logic programming* represent a continued interest in the automation of inference rules.

Such systems can achieve hundreds of thousands of inferences per second, but these inferences are tiny and computation oriented.

Universal methods failed to capture universal solutions:

- In model checking, the state explosion problem limits quickly potential solutions.
- In logic programming, logic specifications are seldom "good" algorithms.

The rise of interactive provers

This early phase of automation was partially supplanted by *interactive* proof environments: Nqthm, Mizar, Coq, NuPRL, HOL, PVS, Matita, etc.

Most of these systems chose either classical, first-order logic with induction or an intuitionistic, higher-order logic as a suitable framework for encoding mathematics.

Many systems were influenced directly or indirectly by Automath and LCF.

Specialization and fracturing

Paths to successful applications of computational logic system require specialization.

- *Pick a domain and specialize.* This approach has lead a number of highly successful systems: SAT solvers, deductive databases, type systems, static analysis, logic circuits, etc.
- *Work within frameworks.* Coq and Isabelle (the "next 700 theorem provers").

What we now refer to as *computational logic* is a *highly fragmented topic:* both as an applied subject and as a theoretical subject.

The cost of fragmentation is high

This fracturing of logic comes with a high cost to the discipline and it greatly diminishes its potential.

- Theoretical and implementation results in one slice are seldom exported to other slices.
- Tactics, libraries, and packages for one framework may not transfer to other frameworks.
- People may view their sub-domain as being universal: missing features are re-invented even if they exist in another slice.
- What should we teach?

A Methodology for Unity

Step 1: Drop mathematics as an intermediate

The traditional approach to reasoning about computation in today's ambitious frameworks follow a two step approach.

1: *Implement mathematics* by picking a general, well understood formal system such as first-order logic, set theory, or a higher-order (constructive) logic.

2: *Reduce computation to mathematics.* Encode computation via some model theoretic (denotational) semantics or as an inductive definition over an operational semantics.

Mathematics does not provide canonical solutions to many intensional aspects of computation: *e.g.*, algorithms, bindings, resources.

A Methodology for Unity

Step 2: Use proof theory as an alternative framework

The proof theory of the *sequent calculus*, in particular, can provide a unifying framework for computational logic.

- Proof theory is more intimately related to computation given its reliance on (mostly) finitary methods.
- Proof theory provides elegant treatments of various intensional aspects of computation (for example, resources in linear logic and binders in terms).
- Proof theory provides rich and flexible avenues of representing and reasoning about computation.

A Methodology for Unity

Step 3: Develop a big logic with many sublogics

- Combine classical, intuitionistic, and linear logics.
- Allow higher-order, first-order, propositional subsets.
- Allow undefined (atomic) and defined (least and greatest fixed points) formulas.

Sequent calculi with cut-elimination theorems allows us to view a wide range of connectives as fitting together orthogonally.

Consistency and "conservative extension" theorems are usually trivial consequences of cut-elimination.

A Methodology for Unity

Step 4: Exploit focused proof systems

Focused proof systems allow one to build macro-level inference rules from micro-level (introduction rules). The resulting macro-rules also satisfy the cut-elimination theorem.

- Focusing requires assigning *polarity* to subformulas. Different choices yield different inference systems: natural deduction, tableaux, free deduction, etc.
- Via fixed points, computations can be incorporated *inside* macro-rules.
- We can engineer a wide range of proof systems.
- Unfocused proofs are just focused proofs with lots of "delays".

Unity must stress a common denominator

Unity should embrace

- *concurrency,* which includes sequentiality,
- *relations,* which includes functions, and
- *non-determinism,* which includes determinism.

This view of unity does not embrace **denotational semantics** nor the **Curry-Howard correspondence** (a.k.a. proofs-as-programs). These seem more related to deep results about particular formalisms.

The **proof search** approach to computational specification more directly supports these elements than does the **proof normalization** approach.

Unity must not be trivial

Some results, some methodology, some discipline must be available to bring all these results together.

Proof theory and cut-elimination provides

- a common language for formulas and proofs;
- duality between abstraction/hiding (*e.g.*, binding) and "re-implementation" (*e.g.*, substitution); and
- orthogonality of logical connectives and features; and
- consistency and conservative extension.

Completeness of focused proof systems is another broadly applicable result.

A primer for proof theory

Structural rules:

Contraction: $\dfrac{\Gamma, B, B \vdash \Delta}{\Gamma, B \vdash \Delta}$ $\dfrac{\Gamma \vdash \Delta, B, B}{\Gamma \vdash \Delta, B}$

Weakening: $\dfrac{\Gamma \vdash \Delta}{\Gamma, B \vdash \Delta}$ $\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, B}$

Identity rules:

$\dfrac{}{B \vdash B}$ Initial $\qquad \dfrac{\Gamma_1 \vdash \Delta_1, B \quad \Gamma_2, B \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}$ Cut

Introduction rules:

$$\frac{\Gamma, B_1, B_2 \vdash \Delta}{\Gamma, B_1 \wedge B_2 \vdash \Delta} \qquad \frac{\Gamma_1 \vdash \Delta_1, B_1 \qquad \Gamma_2 \vdash \Delta_2, B_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, B_1 \wedge B_2}$$

$$\frac{\Gamma, B[t/x] \vdash \Delta}{\Gamma, \forall x B \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta, B[y/x]}{\Gamma \vdash \Delta, \forall x B}$$

Sequents, structural rules, and contexts

Gentzen invented the sequent calculus to solve a problem in the unity of logic.

- He wanted cut-elimination to work for both classical and intuitionistic logic.
- His attempts to prove the *Hauptsatz* using natural deduction failed and lead him to the sequent calculus.
- Remarkably, the difference between classical and intuitionistic logic appeared as a restriction on *structural rules* (no contraction on the right-hand-side).

With the introduction of *linear logic*, Girard has expanded on these themes of using the sequent calculus, the structural rules, and cut-elimination.

Our proposal for a unity of logic continues this story line with a new ingredient: focused proof systems.

Focused proof systems

Invertible introduction inference rules

$$\frac{\Gamma, B_1, B_2 \vdash \Delta}{\Gamma, B_1 \wedge B_2 \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta, B[y/x]}{\Gamma \vdash \Delta, \forall x B}$$

Non-invertible introduction rules

$$\frac{\Gamma, B[t/x] \vdash \Delta}{\Gamma, \forall x B \vdash \Delta} \qquad \frac{\Gamma_1 \vdash \Delta_1, B_1 \qquad \Gamma_2 \vdash \Delta_2, B_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, B_1 \wedge B_2}$$

Focused proofs are built in two phases: one with only invertible rules (the "negative" or "asynchronous" phase); one with only non-invertible rules (the "positive" or "synchronous" phase).

The LKF Focused proof systems for classical logic

$$\frac{\vdash \Theta, C \Uparrow \Gamma}{\vdash \Theta \Uparrow \Gamma, C}\ \mathit{Store} \qquad \frac{\vdash \Theta \Uparrow N}{\vdash \Theta \Downarrow N}\ \mathit{Release}$$

$$\frac{\vdash P, \Theta \Downarrow P}{\vdash P, \Theta \Uparrow \cdot}\ \mathit{Focus} \qquad \frac{}{\vdash \neg P, \Theta \Downarrow P}\ \mathit{Id}\ (\text{literal } P)$$

$$\frac{}{\vdash \Theta \Uparrow \Gamma, t^-} \qquad \frac{\vdash \Theta \Uparrow \Gamma, A \quad \vdash \Theta \Uparrow \Gamma, B}{\vdash \Theta \Uparrow \Gamma, A \wedge^- B}$$

$$\frac{\vdash \Theta \Uparrow \Gamma}{\vdash \Theta \Uparrow \Gamma, f^-} \qquad \frac{\vdash \Theta \Uparrow \Gamma, A, B}{\vdash \Theta \Uparrow \Gamma, A \vee^- B} \qquad \frac{\vdash \Theta \Uparrow \Gamma, A[y/x]}{\vdash \Theta \Uparrow \Gamma, \forall x A}$$

$$\frac{}{\vdash \Theta \Downarrow t^+} \qquad \frac{\vdash \Theta \Downarrow A \quad \vdash \Theta \Downarrow B}{\vdash \Theta \Downarrow A \wedge^+ B} \qquad \frac{\vdash \Theta \Downarrow A_i}{\vdash \Theta \Downarrow A_1 \vee^+ A_2} \qquad \frac{\vdash \Theta \Downarrow A[t/x]}{\vdash \Theta \Downarrow \exists x A}$$

P is positive, N is negative, C is a positive formula or a negative literal, Θ is a multiset of C's

Fixed point μ and equality $=$ as connectives

$$\frac{}{\vdash \Theta \Downarrow t = t} \qquad \frac{}{\vdash \Theta \Uparrow \Gamma, s \neq t}\ \ddagger \qquad \frac{\vdash \Theta\sigma \Uparrow \Gamma\sigma}{\vdash \Theta \Uparrow \Gamma, s \neq t}\ \dagger$$

$\ddagger$ s and t are not unifiable.

$\dagger$ s and t to be unifiable and σ to be their mgu

$$\frac{\vdash \Theta \Uparrow \Gamma, B(\mu B)\bar{t}}{\vdash \Theta \Uparrow \Gamma, \mu B\bar{t}} \qquad \frac{\vdash \Theta \Downarrow B(\mu B)\bar{t}}{\vdash \Theta \Downarrow \mu B\bar{t}}$$

B is a formula with $n \geq 0$ variables abstracted; $\bar{t}$ is a list of n terms.

Here, μ denotes neither the least nor the greatest fixed point.

These distinction arise if we add induction and co-induction.

Examples of fixed points

Natural numbers: terms over 0 for zero and s for successor. Two ways to define predicates over numbers.

$$\begin{aligned} nat\ 0 &\ \texttt{:-}\ true. \\ nat\ (s\ X) &\ \texttt{:-}\ nat\ X. \\ leq\ 0\ Y &\ \texttt{:-}\ true. \\ leq\ (s\ X)\ (s\ Y) &\ \texttt{:-}\ leq\ X\ Y. \end{aligned}$$

Above, as a logic program and below, as fixed points.

$$nat = \mu(\lambda p \lambda x.(x = 0) \vee^{+} \exists y.(s\ y) = x \wedge^{+} p\ y)$$

$$leq = \mu(\lambda q \lambda x \lambda y.(x = 0) \vee^{+} \exists u \exists v.(s\ u) = x \wedge^{+} (s\ v) = y \wedge^{+} q\ u\ v).$$

Horn clauses can be made into fixed point specifications (mutual recursions requires standard encoding techniques).

The engineering of proof systems

Consider proving the positive focused sequent

$$\vdash \Theta \Downarrow (leq\ m\ n \wedge^+ N_1) \vee^+ (leq\ n\ m \wedge^+ N_2),$$

where m, n are natural numbers and N_1, N_2 are negative formulas.

There are exactly two possible macro rules:

$$\frac{\vdash \Theta \Downarrow N_1}{\vdash \Theta \Downarrow (leq\ m\ n \wedge^+ N_1) \vee^+ (leq\ n\ m \wedge^+ N_2)} \text{ for } m \leq n$$

$$\frac{\vdash \Theta \Downarrow N_2}{\vdash \Theta \Downarrow (leq\ m\ n \wedge^+ N_1) \vee^+ (leq\ n\ m \wedge^+ N_2)} \text{ for } n \leq m$$

A macro inference rule can contain an entire Prolog-style computation.

The engineering of proof systems (cont)

Consider proofs involving simulation.

$$sim\ P\ Q \ \equiv\ \forall P' \forall A[\ P \xrightarrow{A} P' \supset \exists Q'\ [Q \xrightarrow{A} Q' \wedge sim\ P'\ Q']].$$

Typically, $P \xrightarrow{A} P'$ is given as a table or as a recursion on syntax (*e.g.*, CCS): hence, as a fixed point.

The body of this expression is exactly two "macro connectives".

- $\forall P' \forall A[P \xrightarrow{A} P' \supset \cdot\ ]$ is a negative "macro connective". There are no choices in expanding this macro rule.
- $\exists Q'[Q \xrightarrow{A} Q' \wedge^{+} \cdot\ ]$ is a positive "macro connective". There can be choices for continuation Q'.

These macro-rules now match exactly the sense of simulation.

Benefits: The unity of computational logic systems

We now see a framework where logic programming, model checking, and theorem proving exist together.

- Logic programming is choosing a path in an unfolding of a fixed point ("may" behavior"). Stress is on *programs*.
- Model checking mixes "must" behaviors (considering all cases) with "may". Stress is on *models*.
- Theorem proving additionally allows the use of invariants to characterize all (possibly infinite) unfoldings. Stress is on *tactics*, etc.

We can now view *programs*, *models*, and *tactics* as different sets of macro-level rules on the same set of micro-level rules.

Benefits: Broad spectrum proof certificates

A variety of existing computational logic systems build (implicitly or explicitly) proofs of various sorts: natural deduction ($\lambda\Pi$), sequent calculus, tableaux, DPLL, tables, *etc.*

These proof styles can be described as focused collections of micro-rules.

An interpreter of micro-rules (for roughly linear logic) can then become a simple, trustable, and broad spectrum proof checker.

Proof compression can arise when some proof details are dropped and the proof checker must spend more time searching for a (small) proof.