Finding Unity in Computational Logic

Dale Miller

INRIA-Saclay, France

ISCL: International School on Computational Logic
Bertinoro, 11-15 April 2011

Lecture 1: Some introductory material.

Many roles of logic in computation

Computation-as-model: Computations happens, *i.e.*, states change, communications occur, *etc*. Logic is used to make statements *about* computation. *E.g.*, Hoare triples, modal logics.

Computation-as-deduction: Elements of logic are used to model elements of computation directly.

Many roles of logic in computation

Computation-as-model: Computations happens, *i.e.*, states change, communications occur, *etc*. Logic is used to make statements *about* computation. *E.g.*, Hoare triples, modal logics.

Computation-as-deduction: Elements of logic are used to model elements of computation directly.

Proof normalization. Programs are proofs and computation is proof normalization (λ-conversion, cut-elimination). A foundations for functional programming.

Proof search. Programs are theories and computation is the search for sequent proofs. A foundations for logic programming.

Many logics to address

There are great many "logics" used practice and in research.

Implemented computational logic systems demand selecting one of two logics: *Classical Logic* and *Intuitionistic Logic*.

These two choices covers a large percentage of existing computational systems based on logic.

[Linear logic lies behind these other two logics.]

Many logics to address

There are great many "logics" used practice and in research.

Implemented computational logic systems demand selecting one of two logics: *Classical Logic* and *Intuitionistic Logic*.

These two choices covers a large percentage of existing computational systems based on logic.

[Linear logic lies behind these other two logics.]

The *propositional vs first-order vs higher-order* logic divide is not a problem of unity: HO logic can directly support both propositional logic and first-order logic.

Logics support many activities

We shall use proof search paradigm (and sequent calculus) to model

- computation (*a la* logic programming)
- model checking
- theorem prover

The first two unfold recursive definitions and explore a space by finite unfoldings.

Theorem proving (with induction and co-induction) attempt to prove things hold for a possibly infinite domains.

Being part of a common framework allows, for example, mixing

- computation and deduction, and
- model checking and theorem proving.

There are many proof systems

- Hilbert systems: linear collection of axioms and inference rules

These are not "analytical" nor "algorithmic". What axioms to start with? How does one reverse *modus ponens*?

There are many proof systems

- Hilbert systems: linear collection of axioms and inference rules

These are not “analytical” nor “algorithmic”. What axioms to start with? How does one reverse *modus ponens*?

- natural deduction
- sequent calculus

Gentzen wanted to provide analytic proof systems for *both* classical and intuitionistic logics: this failed for natural deduction but succeeded for the sequent calculus.

There are many proof systems

- Hilbert systems: linear collection of axioms and inference rules

These are not “analytical” nor “algorithmic”. What axioms to start with? How does one reverse *modus ponens*?

- natural deduction
- sequent calculus

Gentzen wanted to provide analytic proof systems for *both* classical and intuitionistic logics: this failed for natural deduction but succeeded for the sequent calculus.

- tableaux
- resolution refutations

Other useful proof systems with simple connections to the sequent calculus.

Overview of these lectures

Lecture 1: introduction, sequent calculus

Lecture 2: classical and intuitionistic logic

Lecture 3: abstract logic programming

Lecture 4: focused proof systems

Lecture 5: inductive definitions

Lecture 6: model checking and inductive theorem proving