CAS 760 Winter 2010

04 Two Practiced-Oriented Logics

William M. Farmer

Department of Computing and Software
McMaster University

5 April 2010

Outline

- Part 1: LUTINS: The Logic of IMPS
- Part 2: Chiron: A Logic Engineered for Practical Use

Part 1

LUTINS

What is IMPS?

- IMPS is an higher-order theorem proving system developed at The MITRE Corporation by W. Farmer, J. Guttman, and J. Thayer.
- Principal goals:
  - Mechanize mathematical reasoning.
  - Be useful to a wide range of people.
- Approach:
  - Support traditional mathematical techniques.
  - Human oriented instead of machine oriented.
- Main application areas:
  - Hardware and software development.
  - Mathematics education.
- Available free with public license (first released in 1993).
  - Written in T and Common Lisp.
  - User interface based on XEmacs.
  - IMPS web site: `http:imps.mcmaster.ca/`.

Distinguishing Characteristics of IMPS

1. Logic that admits partial functions and undefined terms.
   - Closely corresponds to mathematical practice.
2. Proofs that combine deduction and computation.
   - IMPS proof system is eclectic.
   - Computation plays an essential role in IMPS proofs.
3. Little theories method for organizing mathematics.
   - Essential for formalizing large portions of mathematics.

Goals for the IMPS Logic

- Familiarity: 2-valued, classical, predicate logic.
- Expressivity: Higher-order quantification.
- Support for functions:
  - Higher-order and partial functions.
  - λ-notation and λ-conversion.
  - Definite description.
- Simple type system:
  - No explicit polymorphism.
  - Sort system for classifying expressions by their values.

LUTINS, the Logic of IMPS

- Satisfies all the goals for the IMPS logic.
- A version of Church's simple type theory with:
  - Traditional approach to undefinedness.
  - Additional constructors, including a definite description operator.
  - Sort system for classifying expressions by their values.
- Laws of predicate logic are modified slightly.
  - Instantiation and beta-reduction are restricted to defined expressions.
  - Undefined expressions are indiscernible.

Sorts in LUTINS (1/2)

- A sort α is a syntactic object intended to denote a nonempty set D_α of values.
- Sorts serve as subtypes.
  - α is a subsort of β, written $\alpha \ll \beta$, if $D_\alpha \subseteq D_\beta$.
  - Types are the sorts maximal in the partial order $\ll$.
- Hierarchy of sorts:
  - Atomic sorts like **N**, **Z**, **Q**, **R**.
  - Compound sorts of the form $\alpha_1 \times \cdots \times \alpha_n \rightharpoonup \beta$.
- A compound sort $\alpha_1 \times \cdots \times \alpha_n \rightharpoonup \beta$ denotes the set of partial functions from $D_{\alpha_1} \times \cdots \times D_{\alpha_n}$ to D_β.
  - Sorts are covariant with respect $\rightharpoonup$: If $\alpha \ll \alpha'$ and $\beta \ll \beta'$, then $\alpha \rightharpoonup \beta \ll \alpha' \rightharpoonup \beta'$.
- Examples: $\mathbf{Z} \ll \mathbf{R}$ and $(\mathbf{Z} \to \mathbf{Z}) \ll (\mathbf{R} \to \mathbf{R})$.

Sorts in LUTINS (2/2)

- Every expression e is assigned a sort $\sigma(e)$ according to its syntax (regardless of whether it is defined or not).
  - $\sigma(e) = \alpha$ means the value of e is in D_α if e is defined.
- An expression can "reside" in many other sorts.
  - $(e_\alpha \downarrow \beta)$ denotes $\exists\, x : \beta \,.\, x = e_\alpha$.
  - $(e_\alpha \downarrow)$ denotes $(e_\alpha \downarrow \alpha)$.
- As subtypes, sorts are very convenient for expressing mathematics.

Constructors

Propositional	Binders	Other
the-true	forall	apply-operator
the-false	forsome	equality
and	lambda	if
or	iota	is-defined
if-form	iota-p	defined-in
implies		undefined
iff		
not	with	

Quasi-Constructors

- Quasi-constructors are user-definable constructors.
  - Polymorphic (like ordinary constructors).
  - Used as theory-independent constants.
- Implemented as macro/abbreviations.
- Example: Quasi-equality defined by:

$$e_1 \simeq e_2 \equiv (e_1\downarrow \vee e_2\downarrow \supset e_1 = e_2)$$

IMPS Logic References

1. W. M. Farmer, "A partial functions version of Church's simple theory of types", *Journal of Symbolic Logic*, 55:1269–1291, 1990.
2. W. M. Farmer, "A simple type theory with partial functions and subtypes", *Annals of Pure and Applied Logic*, 64:211–240, 1993.
3. W. M. Farmer, "Theory interpretation in simple type theory", in: J. Heering et al., eds., *Higher-Order Algebra, Logic, and Term Rewriting*, *Lecture Notes in Computer Science*, Vol. 816, pp. 96–123, Springer-Verlag, 1994.
4. W. M. Farmer, "Andrews' type system with undefinedness", in: C. Benzmüller, C. Brown, J. Siekmann, and R. Statman, eds., *Reasoning in Simple Type Theory: Festschrift in Honor of Peter B. Andrews on his 70th Birthday*, *Studies in Logic*, pp. 223–242, College Publications, 2008.

Part 2

Chiron

What is Chiron?

- Chiron is a logic based on NBG set theory that is intended to be a practical, general-purpose logic for mechanizing mathematics.
- Major design goals:
  - Based on familiar and well-understood principles.
  - High theoretical and practical expressivity.
- It is a logic with undefinedness.
- It has a type system with a universal type, dependent types, dependent function types, subtypes, and possibly empty types.
- It has a facility for reasoning about the syntax of expressions that employs quotation and evaluation.
- Ungrounded terms are undefined, and ungrounded formulas (like the liar paradox) are false.

Values

- D_{v} is the domain of sets.
  D_{c} is the domain of classes.
  D_{s} is the domain of superclasses.
  - $D_{\mathrm{v}} \subseteq D_{\mathrm{c}} \subseteq D_{\mathrm{s}}$.
- T, F are the truth values.
  - $\mathrm{T}, \mathrm{F} \notin D_{\mathrm{s}}$.
- $\bot$ is the undefined value.
  - $\bot \notin D_{\mathrm{s}} \cup \{\mathrm{T}, \mathrm{F}\}$.
- For $n \geq 0$, an n-ary operation is a total mapping

  $$o : D_1 \times \cdots \times D_n \to D_{n+1}$$

  where D_i is D_s, $D_c \cup \{\bot\}$, or $\{\mathrm{T}, \mathrm{F}\}$ for all i with $1 \leq i \leq n+1$.
  - An operation $o \notin D_{\mathrm{s}} \cup \{\mathrm{T}, \mathrm{F}, \bot\}$.

Expressions

- An expression of Chiron is inductively defined by the following two formation rules:

  Expr-1 (Atomic expression)

  $$\frac{s \in \mathcal{S}}{\mathbf{expr}[s]}$$

  Expr-2 (Compound expression)

  $$\frac{\mathbf{expr}[e_1], \ldots, \mathbf{expr}[e_n]}{\mathbf{expr}[(e_1, \ldots, e_n)]} \text{ where } n \geq 0.$$

- There are four sorts of proper expressions:
  1. Operators denote operations.
  2. Types denotes superclasses.
  3. Terms denote classes and the undefined value.
  4. Formulas denote truth values.
- Improper expressions are expressions that are not proper expressions; they are nondenoting.

Proper Expressions

Proper Expression	Sort
$(\text{op}, o, k_1, \ldots, k_{n+1})$	operator
$(\text{op-app}, (\text{op}, o, k_1, \ldots, k_{n+1}), e_1, \ldots, e_n)$	type/term/formula
(var, x, α)	term
$(\text{type-app}, \alpha, a)$	type
$(\text{dep-fun-type}, (\text{var}, x, \alpha), \beta)$	type
$(\text{fun-app}, f, a)$	term
$(\text{fun-abs}, (\text{var}, x, \alpha), b)$	term
(if, A, b, c)	term
$(\text{exist}, (\text{var}, x, \alpha), B)$	formula
$(\text{def-des}, (\text{var}, x, \alpha), B)$	term
$(\text{indef-des}, (\text{var}, x, \alpha), B)$	term
(quote, e)	term
(eval, a, k)	type/term/formula

Compact Notation

Compact Notation	Official Notation
$(o :: k_1, \ldots, k_{n+1})$	$(\mathsf{op}, o, k_1, \ldots, k_{n+1})$
$O(e_1, \ldots, e_n)$	$(\mathsf{op\text{-}app}, O, e_1, \ldots, e_n)$
$(x : \alpha)$	$(\mathsf{var}, x, \alpha)$
$\alpha(a)$	$(\mathsf{type\text{-}app}, \alpha, a)$
$(\Lambda\, x : \alpha \,.\, \beta)$	$(\mathsf{dep\text{-}fun\text{-}type}, (\mathsf{var}, x, \alpha), \beta)$
$f(a)$	$(\mathsf{fun\text{-}app}, f, a)$
$(\lambda\, x : \alpha \,.\, b)$	$(\mathsf{fun\text{-}abs}, (\mathsf{var}, x, \alpha), b)$
$\mathsf{if}(A, b, c)$	(if, A, b, c)
$(\exists\, x : \alpha \,.\, B)$	$(\mathsf{exist}, (\mathsf{var}, x, \alpha), B)$
$(\iota\, x : \alpha \,.\, B)$	$(\mathsf{def\text{-}des}, (\mathsf{var}, x, \alpha), B)$
$(\epsilon\, x : \alpha \,.\, B)$	$(\mathsf{indef\text{-}des}, (\mathsf{var}, x, \alpha), B)$
$\lceil e \rceil$	(quote, e)
$[\![a]\!]_k$	(eval, a, k)

Additional Compact Notation

Compact Notation	Defining Expression
T	(true :: formula)( )
F	(false :: formula)( )
V	(set :: type)( )
C	(class :: type)( )
E	(expr :: type)( )
$(a \in b)$	(in :: V, C, formula)(a, b)
$(a =_\alpha b)$	(term-equal :: C, C, type, formula)(a, b, α)
$(a = b)$	$(a =_\mathsf{C} b)$
$(\neg A)$	(not :: formula, formula)(A)
$(A \vee B)$	(or :: formula, formula, formula)(A, B)
$(\forall\, x : \alpha \,.\, A)$	$(\neg(\exists\, x : \alpha \,.\, (\neg A)))$

Constructions

- A construction is a set that represents the construction of an expression.
- (quote, e) denotes the construction that represents the expression e.
- A proper expression e has two different meanings:
  1. The semantic value of e is the value denoted by e itself.
  2. The syntactic value of e is the construction denoted by (quote, e).
- (eval, a, type) denotes the value of the type that is represented by the construction denoted by the term a.
- (eval, a, α) denotes the value of the term of type α that is represented by the construction denoted by the term a.
- (eval, a, formula) denotes the value of the formula that is represented by the construction denoted by the term a.

Example: Differentiation Rule

Meaning formula as an informal formula schema:

$$\text{derivative}(\lambda\, x : \text{real} \;.\; E) = (\lambda\, x : \text{real} \;.\; \text{diff}(E))$$

where E is a rational polynomial.

Meaning formula in Chiron:

$$\forall\, e : \text{rational-polynomials} \;.\; \text{derivative}(\lambda\, x : \text{real} \;.\; [\![e]\!]_{\text{real}}) = (\lambda\, x : \text{real} \;.\; [\![\text{diff}(e)]\!]_{\text{real}})$$

An instance of the meaning formula:

$$\text{derivative}(\lambda\, x : \text{real} \;.\; [\![\ulcorner x^2 + 3x \urcorner]\!]_{\text{real}}) = (\lambda\, x : \text{real} \;.\; [\![\text{diff}(\ulcorner x^2 + 3x \urcorner)]\!]_{\text{real}})$$

which reduces to

$$\text{derivative}(\lambda\, x : \text{real} \;.\; x^2 + 3x) = (\lambda\, x : \text{real} \;.\; 2x + 3)$$

Example: Law of Beta Reduction

Expressed as an informal formula schema:

$$(\lambda\, x : \alpha \;.\; b)(a) \simeq b[(x : \alpha) \mapsto a]$$

where a is free for $(x : \alpha)$ in b.

Expressed as a single formula in Chiron:

$$\begin{array}{l}\forall\, e : \mathsf{E}_{\mathrm{te}} \;. \\ \quad (\text{is-redex}(e) \wedge \\ \quad\; \text{free-for}(\text{redex-arg}(e), \text{redex-var}(e), \text{redex-body}(e))) \\ \quad\quad \supset \\ \quad [\![e]\!]_{\mathrm{te}} \simeq [\![\text{sub}(\text{redex-arg}(e), \text{redex-var}(e), \text{redex-body}(e))]\!]_{\mathrm{te}}\end{array}$$

Chiron References

1. W. M. Farmer, *Chiron: A Set Theory with Types, Undefinedness, Quotation, and Evaluation*, SQRL Report No. 38, 132 pp., McMaster University, 2007 (revised 2010).
2. W. M. Farmer, "Biform theories in Chiron", in: M. Kauers, M. Kerber, R. R. Miner, and W. Windsteiger, eds., *Towards Mechanized Mathematical Assistants*, *Lecture Notes in Computer Science*, Vol. 4573, pp. 66–79, Springer-Verlag, 2007.
3. W. M. Farmer, "Chiron: A multi-paradigm logic", in: R. Matuszewski and A. Zalewska, eds., *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*, *Studies in Logic, Grammar and Rhetoric*, 10(23):1–19, 2007.