Computer Supported Modeling and Reasoning

David Basin, Achim D. Brucker, Jan-Georg Smaus, and Burkhart Wolff

April 2005
http://www.infsec.ethz.ch/education/permanent/csmr/

Metatheory I: Syntax

David Basin

ETH Zurich
8.11.04

Overview

- We have studied reasoning in given theories

  Labs used predeveloped .thy files.

- How does one encode their own theories? Issues include:
  - Metalogic: formalism for formalizing theories
  - Pragmatics: how to use such a metalogic

- The next two lectures will examine:
  - Representing syntax using simple types
  - Representing proofs using dependent types

- We will be formal

  Labs will provide practical experience using formal metatheories

What is the Problem?

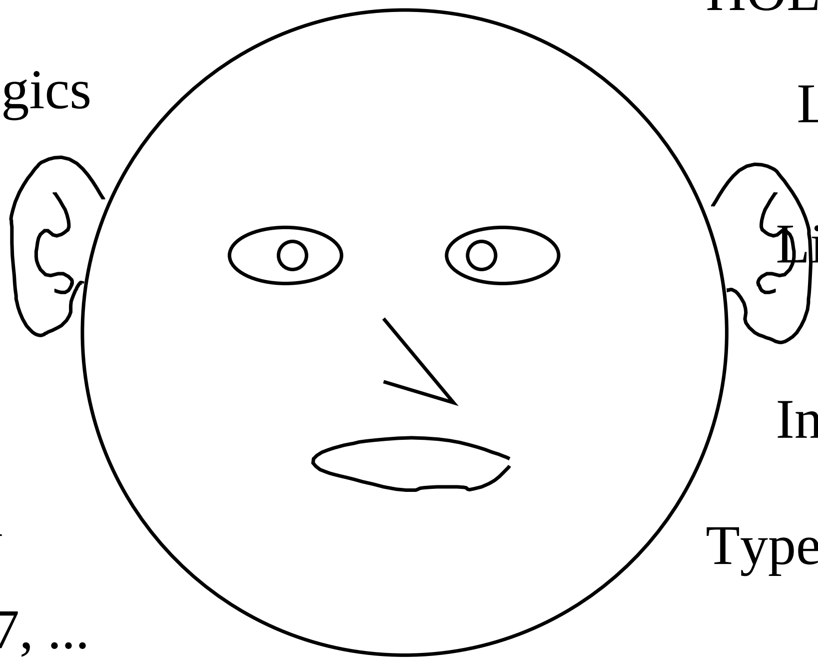
?????

HOL

Non−monotonic Logics

Lambda−calculus

FOL



Linear Logic

Hoare Logic

Intuitionistic Logic

PRA

Type Theory

K, T, S4, S5, S257, ...

Hilbert Presentations, Natural Deduction, Sequent Calculus, ...

Solutions?

- Implement individually

  +/− employment for thousands !

- Embed in a framework logic

  \+ Implement 'core' only once

  \+ Shared support for automation

  \+ Conceptual framework for exploring what a logic is

  +/− Meta-layer between user and logic

  − Makes assumptions about structure of logic

Overview — Syntactic Encodings in Type Theory

- The λ-Calculus as programming language

$$f(x) = g(x, 3) \quad \rightsquigarrow \quad f = \lambda x.\, g\, x\, 3$$

- Simple types classify syntax (o = type of Propositions)

$$\begin{array}{llll} \bot & \rightsquigarrow & False & \in & o \\ \wedge & \rightsquigarrow & And & \in & o \to o \to o \\ \forall & \rightsquigarrow & All & \in & (i \to o) \to o \end{array}$$

- Dependent types classify rules: pr:$o \to Type$

$$\frac{A \wedge B}{A} \quad \rightsquigarrow \quad andel \in \Pi x : o.\, \Pi y : o.\, pr(and\ x\ y) \to pr(x)$$

Overview (cont.)

- Judgments as Types (syntax in this lecture)

$$\begin{array}{c} \vdots\ P \\ \vdash \phi \end{array} \quad \rightsquigarrow \quad \ulcorner P \urcorner \in pr(\ulcorner \phi \urcorner)$$

  - Models syntax: $\phi \in Prop$ *iff* $\ulcorner \phi \urcorner \in o$
  - Models provability: $\vdash_L \phi$ *iff* $\vdash_{TT} pr(\ulcorner \phi \urcorner)$
  - Models proofs: P *iff* $\ulcorner P \urcorner$

- Correctness of encodings: faithfulness and adequacy

  Requires study of metatheory of metalogic: Are our encodings of FOL in $\lambda^{\rightarrow}$ more than just a syntactic trick?

First-Order Syntax with $\lambda^{\rightarrow}$

- Propositional logic

$$P ::= x \mid \neg P \mid P \wedge P \mid P \Rightarrow P \ldots$$

- Programming languages/algebraic specification

datatype *Prop* = *VarInject* of *Variable* | *not* of *Prop*
| *and* of *Prop**Prop | *imp* of *Prop**Prop

- $\lambda^{\rightarrow}$ approach
  - Type declarations for context $\mathcal{B} = \{o\}$
  - Signature types constants:

$$\Sigma = \{not : o \rightarrow o, and : o \rightarrow o \rightarrow o, imp : o \rightarrow o \rightarrow o\}$$

  - Context types propositional variables

First-Order Syntax (cont.)

- Example: $a : o \vdash imp(not\, a)a : o$

$$\dfrac{a : o \vdash imp : o \to o \to o \qquad \dfrac{a : o \vdash not : o \to o \quad a : o \vdash a : o}{a : o \vdash not\, a : o}}{\dfrac{a : o \vdash imp(not\, a) : o \to o \qquad\qquad a : o \vdash a : o}{a : o \vdash imp(not\, a)a : o}}$$

- Non example: $a : o \vdash not(imp\, a)a : o$

$$\dfrac{a : o \vdash not : o \to o \qquad \dfrac{a : o \vdash imp : o \to o \to o \quad a : o, \vdash a : o}{a : o \vdash imp\, a : o \to o}}{???}$$

No proof possible! (requires analysis of normal forms)

First-Order Syntax (cont.)

- Desire bijection $\ulcorner \cdot \urcorner : Prop \rightarrow o$
- Part 1: adequacy

$$p \in Prop \textit{ then } \Gamma \vdash \ulcorner p \urcorner : o$$
$$(\neg a) \Rightarrow b \in Prop \textit{ therefore } imp(not\, a)b : o$$

- Formalize mapping $\ulcorner \cdot \urcorner$

$$\begin{array}{lcl} \ulcorner x \urcorner & = & x \text{ for } x \text{ a variable} \\ \ulcorner \neg P \urcorner & = & not \ulcorner P \urcorner \\ \ulcorner P \wedge Q \urcorner & = & and \ulcorner P \urcorner \ulcorner Q \urcorner \end{array}$$

- Formal statement accounts for variables

  if $x \in FV(P) \Rightarrow x : o \in \Delta$ **and if** $P \in Prop$ **then** $\Delta \vdash \ulcorner P \urcorner : o$

- Proof of adequacy by induction on Prop

FOL/Syntactic Bijection (cont.)

- Part 2: faithfulness

$$\Delta \vdash t : o \textit{ then } \ulcorner t \urcorner^{-1} \in Prop$$

- Define $\ulcorner . \urcorner^{-1}$

$$\begin{array}{lcl} \ulcorner x \urcorner^{-1} & = & x \quad \textit{for } x \textit{ a variable} \\ \ulcorner not\, P \urcorner^{-1} & = & \neg \ulcorner P \urcorner^{-1} \\ \ulcorner and\, P\, Q \urcorner^{-1} & = & \ulcorner P \urcorner \wedge \ulcorner Q \urcorner \end{array}$$

- Trivially $\ulcorner \ulcorner p \urcorner \urcorner^{-1} = p$, but what about $\ulcorner \ulcorner t \urcorner^{-1} \urcorner = t$?

  $t = not\,((\lambda x^{o}.\, x)a)$, $t : o$, what is $\ulcorner t \urcorner^{-1}$?

Faithfulness (cont.)

- Problem: too many representatives in $\lambda^{\rightarrow}$, e.g. $\neg a$

$$\dfrac{a : o \vdash not : o \rightarrow o \qquad a : o \vdash a : o}{a : o \vdash not\, a : o}\ app$$

$$\dfrac{a : o \vdash not : o \rightarrow o \qquad \dfrac{\dfrac{a : o, x : o \vdash x : o}{a : o \vdash \lambda x^{o}.\, x : o \rightarrow o}\ abs \qquad a : o \vdash a : o}{a : o \vdash (\lambda x^{o}.\, x)a : o}\ app}{a : o \vdash not\, ((\lambda x^{o}.\, x)a) : o}\ app$$

Faithfulness (cont.)

- If $t : o$, then $t =_{\beta\eta} t'$, for $t' : o$ a *canonical* ($\beta\eta$-long) normal form

$$\begin{array}{lll} not\,((\lambda x.\, x)a) & =_{\beta\eta} & not\, a \\ not & =_{\beta\eta} & \lambda x.\, not\, x \\ imp\,(not\,((\lambda x.\, x)a)) & =_{\beta\eta} & \lambda x.\, imp\,(not\, a)\, x \end{array}$$

- **Theorem**: The encoding $\ulcorner\cdot\urcorner$ is a bijection between propositional formulae with free variables in Δ and canonical terms t', where $\Delta \vdash t' : o$

Faithfulness (cont.)

- **Proof**: Based on normalization

$$\dfrac{\dfrac{x : \sigma \vdash e : \tau}{\vdash \lambda x^{\sigma}.\, e \; : \sigma \rightarrow \tau}\, abs \qquad \vdash e' : \sigma}{\vdash (\lambda x^{\sigma}.\, e)e' \; : \tau}\, app$$

$$\Downarrow$$

$$\vdash e[x \leftarrow e'] : \tau$$

- **Corollary**: $t : o$ then $t =_{\beta\eta} t'$ and $\ulcorner t' \urcorner^{-1} \in Prop$ for some canonical t'

Problems with First-Order Syntax

- What about quantifiers ?

$$all : var \to o \to o \qquad \forall x.\, p \rightsquigarrow all\; x\; p$$

- First-order syntax requires explicit encoding of standard operations
  - binding: x bound in P in $\forall x.\, P \Leftrightarrow x$ bound in P in $all\; x\; P$
  - Substitution for bound variables:

$$\frac{\forall x.\, P_x}{P_t}\ \forall\text{-}E \qquad \dfrac{\dfrac{\forall x.\, x = x}{x = x[x \leftarrow 0]}\ \forall\text{-}E}{0 = 0}\ \text{Substitution}$$

  - Equivalence under bound variable renaming

$$(\forall x.\, P \Leftrightarrow \forall y.\, P[x \leftarrow y])$$

- Each requires explicit ‘programming’

Higher-Order Abstract Syntax (HOAS)

- Example: first-order arithmetic (FOA)

$$\begin{array}{llcl} \textit{Terms} & T & ::= & x \mid 0 \mid sT \mid T + T \mid T \times T \\ \textit{Formulae} & F & ::= & T = T \mid \neg F \mid F \wedge F \mid \ldots \\ & & & \forall x.\, F \mid \exists x.\, F \end{array}$$

- Type declarations for context $\mathcal{B} = \{i, o\}$

- Signature $\Sigma = \Sigma_T \cup \Sigma_P \cup \Sigma_Q$:

$$\begin{array}{lcl} \Sigma_T & = & \{0 : i,\ s : i \to i,\ plus : i \to i \to i,\ times : i \to i \to i\} \\ \Sigma_P & = & \{eq : i \to i \to o,\ not : o \to o,\ and : o \to o \to o,\ \ldots\} \\ \Sigma_Q & = & \{all : (i \to o) \to o,\ exists : (i \to o) \to o\} \end{array}$$

HOAS (cont.)

- Faithfulness/adequacy: terms and formulae represented by (canonical) members of i and o

$$
\begin{array}{lcl}
0 + s0 & \Leftrightarrow & plus\,0\,(s0) \\
\forall x.\, x = x & \Leftrightarrow & all(\lambda x^{i}.\, eq\, x\, x) \\
\forall x.\, \exists y.\, \neg(x + x = y) & \Leftrightarrow & all(\lambda x^{i}.\, exists(\lambda y^{i}.\, not\,(eq\,(plus\, x\, x)\, y)))
\end{array}
$$

- Example derivation

$$
\dfrac{\vdash all : (i \to o) \to o \qquad \dfrac{\dfrac{\dfrac{x : i \vdash eq : i \to i \to o \qquad x : i \vdash x : i}{x : i \vdash eq\, x : i \to o} \qquad x : i \vdash x : i}{x : i \vdash eq\, x\, x : o}}{\vdash \lambda x^{i}.\, eq\, x\, x : i \to o}}{\vdash all(\lambda x^{i}.\, eq\, x\, x) : o}
$$

HOAS — Why Higher Order Syntax?

- *Order*: For type τ written $\tau_1 \rightarrow \ldots \rightarrow \tau_n \rightarrow \tau_0$, right associated, $\tau_0 \in \mathcal{B}$:
  - $Ord(\tau) = 0$ if $\tau \in \mathcal{B}$
  - $Ord(\tau) = 1 + max(Ord(\tau_i))$,
- Term/propositional operators are first-order

$$and : o \rightarrow o \rightarrow o$$

- Variable binding operators are higher-order

$$all : (i \rightarrow o) \rightarrow o$$

- What is order of summation operator $sum : i \rightarrow i \rightarrow (i \rightarrow i) \rightarrow i$?

$$\sum_{x=0}^{n} (x + 2) \quad \rightsquigarrow sum\, 0\, n\, (\lambda x^{i}.\, plus\, x\, (ss0))$$

HOAS — Why Abstract?

- Standard operations on syntax left implicit
  - binding: x bound in P in $\forall x.\,P \Leftrightarrow x$ bound in P in $all(\lambda x^{i}.\,P)$
  - Substitution for bound variables:

$$\frac{\forall x.\,P_x}{P_t}\ \forall\text{-}E \quad \Leftrightarrow \quad \frac{all(P)}{P(t)}\ \forall\text{-}E$$

$$\cfrac{\cfrac{\forall x.\,x = x}{x = x[x \leftarrow 0]}\ \forall\text{-}E}{0 = 0}\ \text{Substitution} \quad \Leftrightarrow \quad \cfrac{\cfrac{all(\lambda x^{i}.\,x = x)}{(\lambda x^{i}.\,x = x)0}\ \forall\text{-}E}{0 = 0}\ \beta\text{-reduction}$$

  - Equivalence under bound variable renaming

$$(\forall x.\,P \Leftrightarrow \forall y.\,P[x \leftarrow y]) \quad \Leftrightarrow \quad all(\lambda x^{i}.\,P) =_{\alpha} all(\lambda y^{i}.\,P[x \leftarrow y])$$

- $\lambda^{\rightarrow}$ implementation supports standard operations on syntax!

Summary of HOAS

Object Language	Meta Language
Syntactic Category Term, Prop	Type Declaration $\{i, o\} \in \mathcal{B}$
Variable x	Metalogic Variable x
Constructor $\wedge$	First-order Constant $and : o \to o \to o$
Binding Operator $\forall$	Second-order Constant $all : (i \to o) \to o$
Meaningful Expressions $a \wedge b \in Prop$	Members of Types $(and\, a\, b) : o$

Can $\lambda^{\to}$ adequately represent proofs?

- Typical rules for $Prop$ are:

$$\frac{A \wedge B}{A}\ \wedge\text{-}EL \quad \frac{A \wedge B}{B}\ \wedge\text{-}ER \quad \frac{A \quad B}{A \wedge B}\ \wedge\text{-}I$$

- Try ML-style typing with $pf \in \mathcal{B}$

$$\begin{array}{l} andel, ander : pf \to pf \\ andi : pf \to pf \to pf \end{array}$$

- Typing is too weak

$$andel(\ldots)(\ldots) : pf \ \textit{then}\ \ ander(\ldots)(\ldots) : pf$$

- Simple typing doesn't express dependencies

  Analogy to sorting: $\lambda x.x : A$ list $\to A$ list

Representing Proofs (cont.)

- Formulation with dependent types

$$pr : o \rightarrow Type \qquad pr(and\, a\, b) : Type$$

- Classify objects in levels: Term $\in$ Types $\in$ Kinds

$$pr \in o \rightarrow Type \in Kind$$

- Explicit quantification over types (new operator Π)

$$\Pi a^{o} b^{o}.\, pr(and\, a\, b) \rightarrow pr(a)$$

- Desired type theory corresponds to minimal logic over $\forall / \Rightarrow$ with ω-order quantification, known as the LF.

Further Reading

- Hindley and Seldin, Introduction to Combinators and λ-Calculus, Cambridge University Press, 1986.
- N.G. de Bruijn, "A Survey of the Project AUTOMATH", in Essays in Combinatory Logic, Lambda Calculus, and Formalism, Academic Press, 1980
- Harper, Honsell, and Plotkin, "A Framework for Defining Logics", JACM, January 1993.
- Avron, Honsell, Mason, Pollack, "Using Typed Lambda-Calculus to Implement Formal Systems on a Machine", JAR, 1992.