

The Curry-Howard Isomorphism

Software Formal Verification

Maria João Frade

Departamento de Informática
Universidade do Minho

2008/2009

Classical *versus* intuitionistic logic

- **Classical logic** is based on the notion of *truth*.
 - ▶ The truth of a statement is “absolute”: statements are either true or false.
 - ▶ Here “false” means the same as “not true”.
 - ▶ $\phi \vee \neg\phi$ must hold no matter what the meaning of ϕ is.
 - ▶ Information contained in the claim $\phi \vee \neg\phi$ is quite limited.
 - ▶ Proofs using the excluded middle law, $\phi \vee \neg\phi$, or the double negation law, $\neg\neg\phi \rightarrow \phi$ (proof by contradiction), are not *constructive*.
- **Intuitionistic (or constructive) logic** is based on the notion of *proof*.
 - ▶ Rejects the guiding principle of “absolute” truth.
 - ▶ ϕ is “true” if we can prove it.
 - ▶ ϕ is “false” if we can show that if we have a proof of ϕ we get a contradiction.
 - ▶ To show “ $\phi \vee \neg\phi$ ” one have to show ϕ or $\neg\phi$. (If neither of these can be shown, then the putative truth of the disjunction has no justification.)

Intuitionistic (or constructive) logic

Judgements about statements are based on the existence of a proof or “construction” of that statement.

Informal constructive semantics of connectives (BHK-interpretation)

- A proof of $\phi \wedge \psi$ is given by presenting a proof of ϕ and a proof of ψ .
- A proof of $\phi \vee \psi$ is given by presenting either a proof of ϕ or a proof of ψ (plus the stipulation that we want to regard the proof presented as evidence for $\phi \vee \psi$).
- A proof $\phi \rightarrow \psi$ is a construction which permits us to transform any proof of ϕ into a proof of ψ .
- Absurdity \perp (contradiction) has no proof; a proof of $\neg\phi$ is a construction which transforms any hypothetical proof of ϕ into a proof of a contradiction.
- A proof of $\forall x. \phi(x)$ is a construction which transforms a proof of $d \in D$ (D the intended range of the variable x) into a proof of $\phi(d)$.
- A proof of $\exists x. \phi(x)$ is given by providing $d \in D$, and a proof of $\phi(d)$.

Some classical tautologies that are **not** intuitionistically valid

$$\phi \vee \neg\phi$$

excluded middle law

$$\neg\neg\phi \rightarrow \phi$$

double negation law

$$((\phi \rightarrow \psi) \rightarrow \phi) \rightarrow \phi$$

Pierce's law

$$(\phi \rightarrow \psi) \vee (\psi \rightarrow \phi)$$

$$(\phi \rightarrow \psi) \rightarrow (\neg\phi \vee \psi)$$

$$\neg(\phi \wedge \psi) \rightarrow (\neg\phi \vee \neg\psi)$$

$$(\neg\phi \rightarrow \psi) \rightarrow (\neg\psi \rightarrow \phi)$$

$$(\neg\phi \rightarrow \neg\psi) \rightarrow (\psi \rightarrow \phi)$$

$$\neg\forall x. \neg\phi(x) \rightarrow \exists x. \phi(x)$$

$$\neg\exists x. \neg\phi(x) \rightarrow \forall x. \phi(x)$$

$$\neg\forall x. \phi(x) \rightarrow \exists x. \neg\phi(x)$$

The constructive independence of the logical connectives contrast with the classical situation.

Semantics of intuitionistic logic

The semantics of intuitionistic logic are rather more complicated than for the classical case. A model theory can be given by

- *Heyting algebras* or,
- *Kripke semantics*.

Proof systems for intuitionistic logic

- A **natural deduction system** for intuitionistic propositional logic or intuitionistic first-order logic are given by the set of rules presented for PL or FOL, respectively, **except** the rule for the elimination of double negation ($\neg\neg E$).
- Traditionally, classical logic is defined by extending intuitionistic logic with the double negation law, the excluded middle law or with Pierce's law.

The Curry-Howard isomorphism

The Curry-Howard isomorphism establishes a correspondence between natural deduction for intuitionistic logic and λ -calculus.

Observe the analogy between the implicational fragment of intuitionistic propositional logic and $\lambda \rightarrow$

$$\frac{\phi \in \Gamma}{\Gamma \vdash \phi} \text{ (assumption)}$$

$$\frac{(x : \phi) \in \Gamma}{\Gamma \vdash x : \phi} \text{ (var)}$$

$$\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \rightarrow \psi} (\rightarrow_I)$$

$$\frac{\Gamma, x : \phi \vdash e : \psi}{\Gamma \vdash (\lambda x : \phi. e) : \phi \rightarrow \psi} \text{ (abs)}$$

$$\frac{\Gamma \vdash \phi \rightarrow \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi} (\rightarrow_E)$$

$$\frac{\Gamma \vdash a : \phi \rightarrow \psi \quad \Gamma \vdash b : \phi}{\Gamma \vdash a b : \psi} \text{ (app)}$$

The Curry-Howard isomorphism

The connection of type theory to logic is via the *proposition-as-types principle* that establishes a precise relation between intuitionistic logic and λ -calculus.

- a proposition A can be seen as a type (the type of its proofs);
- and a proof of A as a term of type A .

Hence: A is provable \iff A is inhabited

Therefore, the formalization of mathematics in type theory becomes

$$\Gamma \vdash t : A$$

which is equivalent to

$$\text{Type}_\Gamma(t) = A$$

Proof checking boils down to *type checking*.

Type-theoretic notions for proof-checking

In the practice of an **interactive proof assistant based on type theory**, the user types in tactics, guiding the proof development system to construct a proof-term. At the end, this term is type checked and the type is compared with the original goal.

In connection to proof checking there are some **decision problems**:

Type Checking Problem (TCP) $\Gamma \vdash t : A$?

Type Synthesis Problem (TSP) $\Gamma \vdash t : ?$

Type Inhabitation Problem (TIP) $\Gamma \vdash ? : A$

TIP is usually undecidable for type theories of interest.

TCP and TSP are decidable for a large class of interesting type theories.

The reliability of machine checked proofs

Why would one believe a system that says it has verified a proof ?

The proof checker should be a *very small program* that can be verified by hand, giving the highest possible reliability to the proof checker.

de Bruijn criterion

A proof assistant satisfies the de Bruijn criterion if it generates proof-objects (of some form) that can be checked by an 'easy' algorithm.

Proof-objects may be large but they are self-evident. This means that a small program can verify them. The program just follows whether locally the correct steps are being made.

Type-theoretic approach to interactive theorem proving

provability of formula A	\iff	inhabitation of type A
proof checking	\iff	type checking
interactive theorem proving	\iff	interactive construction of a term of a given type

So, decidability of type checking is at the core of the type-theoretic approach to theorem proving.

Proof assistants based on type theory

The first systems of proof checking (type checking) based on the propositions-as-types principle were the systems of the AUTOMATH project.

Modern proof assistants, aggregate to the proof checker a proof-development system for helping the user to develop the proofs interactively.

Examples of proof assistants based on type theory:

- **Coq** - based on the [Calculus of Inductive Constructions](#)
- **Lego** - based on the Extended Calculus of Constructions
- **Agda** - based on Martin-Lof's type theory
- **Nuprl** - based on extensional Martin-Lof's type theory

Higher-Order Logic and Type Theory

Higher-order logic and type theory

- Following Church's original definition of higher-order logic, simply typed λ -calculus is used to describe the **language** of HOL.
- Recall the basic *constructive* core (\forall, \Rightarrow) of HOL:

$$\text{(axiom)} \quad \frac{}{\Delta \vdash \phi} \quad \text{if } \phi \in \Delta$$

$$\text{(\Rightarrow}_I\text{)} \quad \frac{\Delta, \phi \vdash \psi}{\Delta \vdash \phi \Rightarrow \psi}$$

$$\text{(\Rightarrow}_E\text{)} \quad \frac{\Delta \vdash \phi \Rightarrow \psi \quad \Delta \vdash \phi}{\Delta \vdash \psi}$$

$$\text{(\forall}_I\text{)} \quad \frac{\Delta \vdash \psi}{\Delta \vdash \forall x : \sigma. \psi} \quad \text{if } x : \sigma \notin \text{FV}(\Delta)$$

$$\text{(\forall}_E\text{)} \quad \frac{\Delta \vdash \forall x : \sigma. \psi}{\Delta \vdash \psi[x := e]} \quad \text{if } e : \sigma$$

$$\text{(conversion)} \quad \frac{\Delta \vdash \psi}{\Delta \vdash \phi} \quad \text{if } \phi =_{\beta} \psi$$

Higher-order logic and type theory

Following the Curry-Howard isomorphism, why not introduce a λ -term notation for proofs?

$$\text{(axiom)} \quad \frac{}{\Delta \vdash_{\Gamma} x : \phi} \quad \text{if } x : \phi \in \Delta$$

$$\text{(\(\Rightarrow\)_I)} \quad \frac{\Delta, x : \phi \vdash_{\Gamma} e : \psi}{\Delta \vdash_{\Gamma} \lambda x : \phi. e : \phi \Rightarrow \psi}$$

$$\text{(\(\Rightarrow\)_E)} \quad \frac{\Delta \vdash_{\Gamma} a : \phi \Rightarrow \psi \quad \Delta \vdash_{\Gamma} b : \phi}{\Delta \vdash_{\Gamma} a b : \psi}$$

$$\text{(\(\forall\)_I)} \quad \frac{\Delta \vdash_{\Gamma, x : \sigma} e : \psi}{\Delta \vdash_{\Gamma} \lambda x : \sigma. e : \forall x : \sigma. \psi} \quad \text{if } x : \sigma \notin \text{FV}(\Delta)$$

$$\text{(\(\forall\)_E)} \quad \frac{\Delta \vdash_{\Gamma} t : \forall x : \sigma. \psi}{\Delta \vdash_{\Gamma} t e : \psi[x := e]} \quad \text{if } \Gamma \vdash e : \sigma$$

$$\text{(conversion)} \quad \frac{\Delta \vdash_{\Gamma} t : \psi}{\Delta \vdash_{\Gamma} t : \phi} \quad \text{if } \phi =_{\beta} \psi$$

Higher-order logic and type theory

Here we have two “levels” of types theories:

- the (simple) type theory describing the **language** of HOL
- the type theory for the **proof-terms** of HOL

These levels can be put together into one type theory: **λ HOL**.

- Instead of having two separate categories of expressions (terms and types) we have a unique category of expressions, which are called *pseudo-terms*.

- **Pseudo-terms** The set \mathcal{T} of pseudo-terms is defined by

$$A, B, M, N ::= \text{Prop} \mid \text{Type} \mid \text{Type}' \mid x \mid M N \mid \lambda x:A.M \mid \Pi x:A. B$$

We assume a countable set of *variables*: x, y, z, \dots

- $\mathcal{S} \stackrel{\text{def}}{=} \{\text{Prop}, \text{Type}, \text{Type}'\}$ is the set of *sorts* (constants that denote the universes of the type system). We let s range over \mathcal{S} .
- Both Π and λ bind variables. We have the usual notation for free and bound variables.
- Both \Rightarrow and \forall are generalized by a single construction Π . We write $A \rightarrow B$ instead of $\Pi x:A. B$ whenever $x \notin \text{FV}(B)$.

Contexts and judgments

- *Contexts* are used to declare free variables.
- The set of contexts is given by the abstract syntax: $\Gamma ::= \langle \rangle \mid \Gamma, x : A$
- The *domain* of a context is defined by the clause
$$\text{dom}(x_1 : A_1, \dots, x_n : A_n) = \{x_1, \dots, x_n\}$$
- A *judgment* is a triple of the form $\Gamma \vdash A : B$ where $A, B \in \mathcal{T}$ and Γ is a context.
- A judgment is *derivable* if it can be inferred from the typing rules of next slide.
 - ▶ If $\Gamma \vdash A : B$ then Γ , A and B are *legal*.
 - ▶ If $\Gamma \vdash A : s$ for $s \in \mathcal{S}$ we say that A is a *type*.

The typing rules are *parametrized*.

λ HOL - typing rules

(axioms)	$\langle \rangle \vdash \text{Prop} : \text{Type}$	$\langle \rangle \vdash \text{Type} : \text{Type}'$
(var)	$\frac{\Gamma \vdash A : s}{\Gamma, x:A \vdash x : A}$	if $x \notin \text{dom}(\Gamma)$
(weak)	$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma, x:B \vdash M : A}$	if $x \notin \text{dom}(\Gamma)$
(Π)	$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x:A \vdash B : s_2}{\Gamma \vdash (\Pi x:A. B) : s_2}$	if $(s_1, s_2) \in \{(\text{Type}, \text{Type}), (\text{Prop}, \text{Prop}), (\text{Type}, \text{Prop})\}$
(app)	$\frac{\Gamma \vdash M : (\Pi x:A. B) \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]}$	
(λ)	$\frac{\Gamma, x:A \vdash M : B \quad \Gamma \vdash (\Pi x:A. B) : s}{\Gamma \vdash \lambda x:A. M : (\Pi x:A. B)}$	
(conv)	$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma \vdash M : B}$	if $A =_{\beta} B$

$$(II) \quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x:A \vdash B : s_2}{\Gamma \vdash (\Pi x:A. B) : s_2} \quad \text{if } (s_1, s_2) \in \{(\text{Type}, \text{Type}), (\text{Prop}, \text{Prop}), (\text{Type}, \text{Prop})\}$$

- **(Type, Type)** forms the function type $A \rightarrow B$ for $A : \text{Type}$ and $B : \text{Type}$; **predicate types**. This comprises
 - ▶ **unary or binary predicates** like: $A \rightarrow \text{Prop}$ or $A \rightarrow A \rightarrow \text{Prop}$;
 - ▶ **higher-order predicates** like: $(A \rightarrow A \rightarrow \text{Prop}) \rightarrow \text{Prop}$.
- **(Prop, Prop)** forms the propositional type $\phi \rightarrow \psi$ for $\phi : \text{Prop}$ and $\psi : \text{Prop}$; **propositional formulas**.
- **(Type, Prop)** forms the dependent propositional type $(\Pi x:A. \psi)$ for $A : \text{Type}$ and $\psi : \text{Prop}$; **universally quantified formulas**.

Dependent types

Type constructor Π captures in the type theory the set-theoretic notion of *generic* or *dependent function space*.

Dependent functions

The type of this kind of functions is $\Pi x:A. B$, the product of a family $\{B(x)\}_{x:A}$ of types. Intuitively

$$\Pi x:A. B(x) = \left\{ f : A \rightarrow \bigcup_{x:A} B(x) \mid \forall a:A. fa : B(a) \right\}$$

i.e., a type of functions f where the range-set depends on the input value.

If $f : \Pi x:A. B(x)$, then f is a function with domain A and such that $fa : B(a)$ for every $a : A$.

Dependent types

A *dependent type* is a type that may depend on a value, typically like:

- a **predicate**, which depends on its domain. For instance, the predicate even over natural numbers

$$\text{even} : \text{nat} \rightarrow \text{Prop}$$

Universal quantification is a dependent function. For instance, $\forall x : \text{nat}. \text{even } x$ is encoded by

$$\prod x : \text{nat}. \text{even } x$$

- an **array type** (or vector), which depends on its length. For instance, the polymorphic dependent type constructor

$$\text{Vec} : \text{Type} \rightarrow \text{nat} \rightarrow \text{Type}$$

Here is an example of a dependent function in a Haskell like syntax:

$$\begin{aligned} \text{gen} &:: \prod x : \text{nat}. a \rightarrow \text{Vec } a \ x \\ \text{gen } 0 \ x &= [] \\ \text{gen } (n + 1) \ x &= x : (\text{gen } n \ x) \end{aligned}$$

λ HOL - examples

Recall the Leibniz equality. For $A : \text{Type}$, $x : A$, $y : A$,

$$(x =_L y) \stackrel{\text{def}}{=} \prod P : A \rightarrow \text{Prop}. Px \rightarrow Py$$

Let $\Gamma \stackrel{\text{def}}{=} A : \text{Type}, x : A$

Reflexivity $A : \text{Type}, x : A \vdash (\lambda P : A \rightarrow \text{Prop}. \lambda q : Px. q) : (x =_L x)$

$$\frac{\frac{\frac{}{(3)}}{\Gamma, P : A \rightarrow \text{Prop} \vdash Px : \text{Prop}}{\Gamma, P : A \rightarrow \text{Prop}, q : Px \vdash q : Px} \text{ (var)}}{\Gamma, P : A \rightarrow \text{Prop} \vdash \lambda q : Px. q : Px \rightarrow Px} \text{ (2)}}{\Gamma \vdash (\lambda P : A \rightarrow \text{Prop}. \lambda q : Px. q) : (x =_L x) : \text{Prop}} \text{ (\lambda)} \text{ (1)}$$

(1)

$$\frac{\frac{\frac{}{(4)}}{\Gamma \vdash A \rightarrow \text{Prop} : \text{Type}}{\Gamma, P : A \rightarrow \text{Prop} \vdash A \rightarrow \text{Prop} : \text{Type}} \text{ (4)}}{\Gamma \vdash \prod P : A \rightarrow \text{Prop}. Px \rightarrow Px : \text{Prop}} \text{ (weak)} \text{ (2)}}{\Gamma \vdash \prod P : A \rightarrow \text{Prop}. Px \rightarrow Px : \text{Prop}} \text{ (\Pi)}$$

λ HOL - examples

(2)

$$\frac{\frac{\frac{}{\Gamma, P : A \rightarrow \text{Prop} \vdash Px : \text{Prop}} \text{(3)}}{\Gamma, P : A \rightarrow \text{Prop} \vdash Px : \text{Prop}} \text{(3)} \quad \frac{\frac{\frac{}{\Gamma, P : A \rightarrow \text{Prop} \vdash Px : \text{Prop}} \text{(3)}}{\Gamma, P : A \rightarrow \text{Prop} \vdash Px : \text{Prop}} \text{(3)}}{\Gamma, P : A \rightarrow \text{Prop}, z : Px \vdash Px : \text{Prop}} \text{(weak)}}{\Gamma, P : A \rightarrow \text{Prop} \vdash Px \rightarrow Px : \text{Prop}} \text{(II)}$$

(3)

$$\frac{\frac{\frac{}{\Gamma \vdash A \rightarrow \text{Prop} : \text{Type}} \text{(4)}}{\Gamma, P : A \rightarrow \text{Prop} \vdash P : A \rightarrow \text{Prop}} \text{(var)} \quad \frac{\frac{\frac{\frac{}{\vdash \text{Type} : \text{Type}'} \text{(axiom)}}{A : \text{Type} \vdash A : \text{Type}} \text{(var)}}{\Gamma \vdash x : A} \text{(var)}}{\Gamma, P : A \rightarrow \text{Prop} \vdash x : A} \text{(weak)}}{\Gamma, P : A \rightarrow \text{Prop} \vdash Px : \text{Prop}} \text{(app)}$$

λ HOL - examples

(4)

$$\frac{\frac{\frac{\frac{\Gamma \vdash A : \text{Type}}{\Gamma \vdash A : \text{Type}} \text{ (5)}}{\vdash \text{Prop} : \text{Type}} \text{ (axiom)}}{\vdash \text{Type} : \text{Type}'} \text{ (axiom)}}{A : \text{Type} \vdash \text{Prop} : \text{Type}} \text{ (weak)}}{\Gamma \vdash \text{Prop} : \text{Type}} \text{ (weak)} \quad \frac{\frac{\Gamma \vdash A : \text{Type}}{\Gamma \vdash A : \text{Type}} \text{ (5)}}{\Gamma, z : A \vdash \text{Prop} : \text{Type}} \text{ (weak)}}{\Gamma \vdash A \rightarrow \text{Prop} : \text{Type}} \text{ (II)}$$

(5)

$$\frac{\frac{\frac{\vdash \text{Type} : \text{Type}'}{A : \text{Type} \vdash A : \text{Type}} \text{ (var)}}{\vdash \text{Type} : \text{Type}'} \text{ (axiom)}}{A : \text{Type} \vdash A : \text{Type}} \text{ (weak)} \quad \frac{\frac{\vdash \text{Type} : \text{Type}'}{A : \text{Type} \vdash A : \text{Type}} \text{ (var)}}{A : \text{Type} \vdash A : \text{Type}} \text{ (weak)}}{\Gamma \vdash A : \text{Type}} \text{ (weak)}$$

λ HOL - examples

Recall the Leibniz equality. For $A : \text{Type}$, $x : A$, $y : A$,

$$(x =_L y) \stackrel{\text{def}}{=} \prod P : A \rightarrow \text{Prop}. Px \rightarrow Py$$

Let us now prove symmetry for the Leibniz equality.

Let $\Gamma \stackrel{\text{def}}{=} A : \text{Type}, x : A, y : A, t : (x =_L y)$

Symmetry $\Gamma \vdash t(\lambda z : A. z =_L x)(\lambda P : A \rightarrow \text{Prop}. \lambda q : Px. q) : (y =_L x)$

$$\frac{\frac{\frac{\vdots}{\Gamma \vdash t : (x =_L y)}}{\Gamma \vdash t(\lambda z : A. z =_L x)} \quad \frac{\frac{\vdots}{\Gamma \vdash (\lambda z : A. z =_L x) : A \rightarrow \text{Prop}}}{\Gamma \vdash (\lambda z : A. z =_L x)x \rightarrow (\lambda z : A. z =_L x)y} \quad \vdots}{\Gamma \vdash t(\lambda z : A. z =_L x) : (x =_L x) \rightarrow (y =_L x)} \text{ (conv)} \quad \frac{\vdots}{\Gamma \vdash w : (x =_L x)}}{\Gamma \vdash t(\lambda z : A. z =_L x)w : (y =_L x)}$$

where w is the proof-term of reflexivity $(\lambda P : X \rightarrow \text{Prop}. \lambda q : Px. q)$

Properties of $\lambda\mathbf{HOL}$

There is a **formulas-as-types isomorphism** between intuitionistic HOL and $\lambda\mathbf{HOL}$

Uniqueness of types

If $\Gamma \vdash M : A$ and $\Gamma \vdash M : B$, then $A =_{\beta} B$.

Subject reduction

If $\Gamma \vdash M : A$ and $M \rightarrow_{\beta} N$, then $\Gamma \vdash N : A$.

Strong normalization

If $\Gamma \vdash M : A$, then all β -reductions from M terminate.

Confluence

If $M =_{\beta} N$, then $M \rightarrow_{\beta} R$ and $N \rightarrow_{\beta} R$, for some term R .

Properties of $\lambda\mathbf{HOL}$

Recall the decidability problems:

Type Checking Problem (TCP)	$\Gamma \vdash M : A$?
Type Synthesis Problem (TSP)	$\Gamma \vdash M :$?
Type Inhabitation Problem (TIP)	$\Gamma \vdash$? : A

For $\lambda\mathbf{HOL}$:

- TIP is undecidable.
- TCP and TSP are decidable.

Remark

Normalization and type checking are intimately connected due to (conv) rule.

Deciding equality of dependent types, and hence deciding the well-typedness of a dependent typed terms, requires to perform computations. If non-normalizing terms are allowed in types, then TCP and TSP become undecidable.

Direct encoding.

- Each logical construction have a counterpart in the type theory.
- Theorem proving consists of the (interactive) construction of a **proof-term**, which can be easily checked independently.
- Examples:
 - ▶ **Coq** - based on the Calculus of Inductive Constructions
 - ▶ **Agda** - based on Martin-Lof's type theory
 - ▶ **Lego** - based on the Extended Calculus of Constructions
 - ▶ **Nuprl** - based on extensional Martin-Lof's type theory

Encoding of logic in type theory

Shallow encoding (*Logical Frameworks*).

- The type theory is used as a logical framework, a meta system for encoding a specific logic one wants to work with.
- The encoding of a logic L is done by choosing an appropriate context Γ_L , in which the language of L and the proof rules are declared.
- Usually, the proof-assistants based on this kind of encoding **do not produce standard proof-objects**, just *proof-scripts*.
- Examples:
 - ▶ **HOL**, based on the Church's simple type theory. This is a classical higher-order logic.
 - ▶ **Isabelle**, based on intuitionistic simple type theory (used as the meta logic). Various logics (FOL, HOL, sequent calculi,...) are described.