Deductive Reasoning and the Coq Proof Assistant

Maria João Frade

HASLab - INESC TEC
Departamento de Informática, Universidade do Minho

2018/2019

Roadmap

- **Natural Deduction**
  - natural deduction proof system for propositional and predicate logic;
  - forward and backward reasoning;
  - admissible rules; derivable rules;
  - soundness; completeness.
- **Proposition as Types**
  - intuitionistic understanding of logic;
  - the Curry-Howard isomorphism;
  - type-theoretical notions for proof-checking;
  - proof assistants based on type theory.
- **Coq in Brief**
  - main features of the Coq proof-assistant;
  - Coq syntax; declarations and definitions; computation;
  - proof development; some tactics for first-order reasoning.

Natural Deduction

Introduction

- So far we have taken the "semantic" approach to logic. This, however, is not the only possible point of view.
- Instead of adopting the view based on the notion of truth, we can think of logic as a codification of reasoning. This alternative approach to logic, called "deductive", focuses directly on the deduction relation that is induced on formulas.
- A *proof system* (or *inference system*) consists of a set of basic rules for constructing derivations. Such a derivation is a formal object that encodes an explanation of why a given formula – the conclusion – is deducible from a set of assumptions.
- The rules that govern the construction of derivations are called *inference rules* and consist of zero or more *premises* and a single *conclusion*. Derivations have a tree-like shape. We use the standard notation of separating the premises from the conclusion by a horizontal line.

$$\frac{perm_1 \quad \ldots \quad perm_n}{concl}$$

Natural deduction

- The proof system we will present here is a formalisation of the reasoning used in mathematics, and was introduced by Gerhard Gentzen in the first half of the 20th century as a "natural" representation of logical derivations. It is for this reason called *natural deduction*.
- We choose to present the rules of natural deduction in sequent style.
- A *sequent* is a judgment of the form $\Gamma \vdash A$, where Γ is a set of formulas (the *context*) and A a formula (the *conclusion* of the sequent).
- A sequent $\Gamma \vdash A$ is meant to be read as "A can be deduced from the set of *assumptions* Γ", or simply "A is a consequence of Γ".

Natural deduction

The set of basic rules provided is intended to aid the translation of thought (mathematical reasoning) into formal proof.

For example, if F and G can be deduced from Γ, then $F \wedge G$ can also be deduced from Γ .

This is the "$\wedge$-introduction" rule

$$\frac{\Gamma \vdash F \quad \Gamma \vdash G}{\Gamma \vdash F \wedge G}\ \wedge_{\mathsf{I}}$$

There are two "$\wedge$-elimination" rules:

$$\frac{\Gamma \vdash F \wedge G}{\Gamma \vdash F}\ \wedge_{\mathsf{E1}} \qquad\qquad \frac{\Gamma \vdash F \wedge G}{\Gamma \vdash G}\ \wedge_{\mathsf{E2}}$$

Natural deduction

- This system is intended for human use, in the sense that
  - a person can guide the proof process;
  - the proof produced is highly legible, and easy to understand.

  This contrast with decision procedures that just produce a "yes/no" answer, and may not give insight into the relationship between the assumption and the conclusion.
- We present natural deduction in sequent style, because
  - it gives a clear representation of the discharging of assumptions;
  - it is closer to what one gets while developing a proof in a proof-assistant.

Natural deduction for PL

- An *instance* of an inference rule is obtained by replacing all occurrences of each meta-variable by a phrase in its range. An inference rule containing no premises is called an *axiom schema* (or simply, an *axiom*).

The proof system $\mathcal{N}_{\mathsf{PL}}$ of *natural deduction* for propositional logic is defined by the rules presented in the next slide. A *derivation* (or *proof*) in $\mathcal{N}_{\mathsf{PL}}$ is inductively defined by the following clause:

- If
$$\frac{\Gamma_1 \vdash A_1 \quad \ldots \quad \Gamma_n \vdash A_n}{\Gamma \vdash A}\ (\mathsf{R})$$
is an instance of rule (R) of the proof system, and $\mathcal{D}_i$ is a derivation with conclusion $\Gamma_i \vdash A_i$ (for $1 \leq i \leq n$), then
$$\frac{\mathcal{D}_1 \quad \ldots \quad \mathcal{D}_n}{\Gamma \vdash A}\ (\mathsf{R})$$

A sequent $\Gamma \vdash A$ is *derivable* in $\mathcal{N}_{\mathsf{PL}}$ if it is the conclusion of some derivation.

System $\mathcal{N}_{\mathsf{PL}}$ for classical propositional logic

$$\frac{}{\Gamma \vdash \top}\ \text{true} \qquad\qquad \frac{A \in \Gamma}{\Gamma \vdash A}\ \text{assumption}$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge_{\mathsf{E1}} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge_{\mathsf{E2}} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge_{\mathsf{I}}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee_{\mathsf{I1}} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee_{\mathsf{I2}} \qquad \frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\ \vee_{\mathsf{E}}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \to_{\mathsf{I}} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash A \to B}{\Gamma \vdash B}\ \to_{\mathsf{E}}$$

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\ \neg_{\mathsf{I}} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash \bot}\ \neg_{\mathsf{E}}$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \bot_{\mathsf{E}} \qquad \frac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A}\ \text{RAA}$$

Proof presentation

$\vdash \neg P \to (Q \to P) \to \neg Q$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg P, Q \to P, Q \vdash Q} \quad \dfrac{}{\neg P, Q \to P, Q \vdash Q \to P}}{\neg P, Q \to P, Q \vdash P}\ \to_{\mathsf{E}} \quad \dfrac{}{\neg P, Q \to P, Q \vdash \neg P}}{\neg P, Q \to P, Q \vdash \bot}\ \neg_{\mathsf{E}}}{\neg P, Q \to P \vdash \neg Q}\ \neg_{\mathsf{I}}}{\neg P \vdash (Q \to P) \to \neg Q}\ \to_{\mathsf{I}}}{\vdash \neg P \to (Q \to P) \to \neg Q}\ \to_{\mathsf{I}}}{}$$

- This example shows that even for such a reasonably simple formula, the size of the tree already poses a problem from the point of view of its representation.
- For that reason, we shall adopt an alternative format for presenting bigger proof trees.

Backward reasoning

- This presentation style in fact corresponds to a popular strategy for constructing derivations. In *backward reasoning* one starts with the conclusion sequent and chooses to apply a rule that can justify that conclusion; one then repeats the procedure on the resulting premises.

$\vdash \neg P \to (Q \to P) \to \neg Q$

$\vdash \neg P \to (Q \to P) \to \neg Q$	$\to_{\mathsf{I}}$
1. $\neg P \vdash (Q \to P) \to \neg Q$	$\to_{\mathsf{I}}$
1. $\neg P, Q \to P \vdash \neg Q$	$\neg_{\mathsf{I}}$
1. $\neg P, Q \to P, Q \vdash \bot$	$\neg_{\mathsf{E}}$
1. $\neg P, Q \to P, Q \vdash P$	$\to_{\mathsf{E}}$
1. $\neg P, Q \to P, Q \vdash Q$	assumption
2. $\neg P, Q \to P, Q \vdash Q \to P$	assumption
2. $\neg P, Q \to P, Q \vdash \neg P$	assumption

- In a proof-assistant the proof is usually developed backwards.

Forward reasoning

- If one prefers to present derivations in a forward fashion, which corresponds to constructing derivations using the *forward reasoning* strategy, then it is customary to simply give sequences of judgments, each of which is either an axiom or follows from a preceding judgment in the sequence, by an instance of an inference rule.

$\vdash \neg P \to (Q \to P) \to \neg Q$

	Judgment	Justification
1.	$\neg P, Q \to P, Q \vdash Q$	assumption
2.	$\neg P, Q \to P, Q \vdash Q \to P$	assumption
3.	$\neg P, Q \to P, Q \vdash P$	$\to_{\mathsf{E}}$ 1, 2
4.	$\neg P, Q \to P, Q \vdash \neg P$	assumption
5.	$\neg P, Q \to P, Q \vdash \bot$	$\neg_{\mathsf{E}}$ 3, 4
6.	$\neg P, Q \to P \vdash \neg Q$	$\neg_{\mathsf{I}}$ 5
7.	$\neg P \vdash (Q \to P) \to \neg Q$	$\to_{\mathsf{I}}$ 6
8.	$\vdash \neg P \to (Q \to P) \to \neg Q$	$\to_{\mathsf{I}}$ 7

In a proof-assistant

In a proof-assistant, the usual approach is to develop the proof backwards by a method that is known as *goal directed proof*:

1. The user enters a statement that he wants to prove.
2. The system displays the formula as a formula to be proved, possibly giving a context of local facts that can be used for this proof.
3. The user enters a command (a basic rule or a *tactic*) to decompose the goal into simpler ones.
4. The system displays a list of formulas that still need to be proved.

When there are no more goals the proof is complete!

An example

$\vdash A \vee \neg A$ proved in backward direction

$\vdash A \vee \neg A$	RAA
1. $\neg(A \vee \neg A) \vdash \bot$	$\neg_E$
1. $\neg(A \vee \neg A) \vdash \neg(A \vee \neg A)$	assumption
2. $\neg(A \vee \neg A) \vdash A \vee \neg A$	$\vee_{I2}$
1. $\neg(A \vee \neg A) \vdash \neg A$	$\neg_I$
1. $\neg(A \vee \neg A), A \vdash \bot$	$\neg_E$
1. $\neg(A \vee \neg A), A \vdash A \vee \neg A$	$\vee_{I1}$
1. $\neg(A \vee \neg A), A \vdash A$	assumption
2. $\neg(A \vee \neg A), A \vdash \neg(A \vee \neg A)$	assumption

Admissible rule

An inference rule is *admissible* in a formal system if every judgement that can be proved making use of that rule can also be proved without it (in other words the set of judgements of the system is closed under the rule).

Weakening

The following rule, named *weakening*, is admissible in $\mathcal{N}_{\mathsf{PL}}$

$$\frac{\Gamma \vdash A}{\Gamma, B \vdash A}$$

Derivable rule

An inference rule is said to be *derivable* in a proof system if the conclusion of the rule can be derived from its premisses using the other rules of the system.

Example of a derivable rule

	Judgment	Justification
1.	$\Gamma \vdash A \wedge B$	premise
2.	$\Gamma \vdash A$	$\wedge_{E1}$ 1
3.	$\Gamma \vdash B$	$\wedge_{E2}$ 1
4.	$\Gamma \vdash B \wedge A$	$\wedge_I$ 3, 2

Hence the rule $\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash B \wedge A}$ is derivable.

Soundness and completeness of PL

Soundness

If $\Gamma \vdash F$, then $\Gamma \models F$.

Completeness

If $\Gamma \models F$, then $\Gamma \vdash F$.

Natural deduction for FOL

- We present here a natural deduction proof system for classical first-order logic in sequent style.
- Derivations in FOL will be similar to derivations in PL, except that we will have new proof rules for dealing with the quantifiers.
- More precisely, we overload the proof rules of PL, and we add introduction and elimination rules for the quantifiers. This means that the proofs developed for PL still hold in this proof system.

The proof system $\mathcal{N}_{\mathsf{FOL}}$ of natural deduction for first-order logic is defined by the rules presented in the next slide.

- An instance of an inference rule is obtained by replacing all occurrences of each meta-variable by a phrase in its range. In some rules, there may be side conditions that must be satisfied by this replacement. Also, there may be syntactic operations (such as substitutions) that have to be carried out after the replacement.

System $\mathcal{N}_{\mathsf{FOL}}$ for classical first-order logic

$$\frac{}{\Gamma \vdash \top}\ \text{true} \qquad \frac{\phi \in \Gamma}{\Gamma \vdash \phi}\ \text{assumption}$$

$$\frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \phi}\ \wedge_{\mathsf{E1}} \qquad \frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \psi}\ \wedge_{\mathsf{E2}} \qquad \frac{\Gamma \vdash \phi \quad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi}\ \wedge_I$$

$$\frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi}\ \vee_{\mathsf{I1}} \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi}\ \vee_{\mathsf{I2}} \qquad \frac{\Gamma \vdash \phi \vee \psi \quad \Gamma, \phi \vdash \theta \quad \Gamma, \psi \vdash \theta}{\Gamma \vdash \theta}\ \vee_{\mathsf{E}}$$

$$\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \to \psi}\ \to_{\mathsf{I}} \qquad \frac{\Gamma \vdash \phi \quad \Gamma \vdash \phi \to \psi}{\Gamma \vdash \psi}\ \to_{\mathsf{E}}$$

$$\frac{\Gamma, \phi \vdash \bot}{\Gamma \vdash \neg\phi}\ \neg_{\mathsf{I}} \qquad \frac{\Gamma \vdash \phi \quad \Gamma \vdash \neg\phi}{\Gamma \vdash \bot}\ \neg_{\mathsf{E}}$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \phi}\ \bot_{\mathsf{E}} \qquad \frac{\Gamma, \neg\phi \vdash \bot}{\Gamma \vdash \phi}\ \text{RAA}$$

System $\mathcal{N}_{\mathsf{FOL}}$ for classical first-order logic

Proof rules for quantifiers.

$$\frac{\Gamma \vdash \phi[y/x]}{\Gamma \vdash \forall x.\, \phi}\ \forall_{\mathsf{I}}\ \textbf{(a)} \qquad \frac{\Gamma \vdash \forall x.\, \phi}{\Gamma \vdash \phi[t/x]}\ \forall_{\mathsf{E}}$$

$$\frac{\Gamma \vdash \phi[t/x]}{\Gamma \vdash \exists x.\, \phi}\ \exists_{\mathsf{I}} \qquad \frac{\Gamma \vdash \exists x.\, \phi \quad \Gamma, \phi[y/x] \vdash \theta}{\Gamma \vdash \theta}\ \exists_{\mathsf{E}}\ \textbf{(b)}$$

(a) y must not occur free in either Γ or ϕ.

(b) y must not occur free in either Γ, ϕ or θ.

System $\mathcal{N}_{\mathsf{FOL}}$ for classical first-order logic

- Rule $\forall_\mathsf{I}$ tells us that if $\phi[y/x]$ can be deduced from Γ for a variable y that does not occur free in either Γ or ϕ, then $\forall x.\phi$ can also be deduced from Γ because y is fresh. The side condition (a) stating that y must not be free in ϕ or in any formula of Γ is crucial for the soundness of this rule. As y is a fresh variable we can think of it as an indeterminate term, which justifies that $\forall x.\phi$ can be deduced from Γ.
- Rule $\forall_\mathsf{E}$ says that if $\forall x.\phi$ can be deduced from Γ then the x in ϕ can be replaced by any term t assuming that t is free for x in ϕ (this is implicit in the notation). It is easy to understand that this rule is sound: if ϕ is true for all x, then it must be true for any particular term t.
- Rule $\exists_\mathsf{I}$ tells us that if it can be deduced from Γ that $\phi[t/x]$ for some term t which is free for x in ϕ (this proviso is implicit in the notation), then $\exists x.\phi$ can also be deduced from Γ.
- The second premise of rule $\exists_\mathsf{E}$ tells us that θ can be deduced if, additionally to Γ, ϕ holds for an indeterminate term. But the first premise states that such a term exists, thus θ can be deduced from Γ with no further assumptions.

An example

$(\exists x.\neg\psi) \to \neg\forall x.\psi$ is a theorem

$\vdash (\exists x.\neg\psi) \to \neg\forall x.\psi$ — $\to_\mathsf{I}$
- **1.** $\exists x.\neg\psi \vdash \neg\forall x.\psi$ — $\neg_\mathsf{I}$
  - **1.** $\exists x.\neg\psi, \forall x.\psi \vdash \perp$ — $\exists_\mathsf{E}$
    - **1.** $\exists x.\neg\psi, \forall x.\psi \vdash \exists x.\neg\psi$ — assumption
    - **2.** $\exists x.\neg\psi, \forall x.\psi, \neg\psi[x_0/x] \vdash \perp$ — $\neg_\mathsf{E}$
      - **1.** $\exists x.\neg\psi, \forall x.\psi, \neg\psi[x_0/x] \vdash \psi[x_0/x]$ — $\forall_\mathsf{E}$
        - **1.** $\exists x.\neg\psi, \forall x.\psi, \neg\psi[x_0/x] \vdash \forall x.\psi$ — assumption
      - **2.** $\exists x.\neg\psi, \forall x.\psi, \neg\psi[x_0/x] \vdash \neg\psi[x_0/x]$ — assumption

Note that when the rule $\exists_\mathsf{E}$ is applied a fresh variable x_0 is introduced. The side condition imposes that x_0 must not occur free either in $\exists x.\neg\psi$ or in $\forall x.\psi$.

An example

Instead of explicitly write the substitutions, the following derivation adopts the convention to establish the converse implication.

$\phi(x_1, \dots, x_n)$ to denote a formula having free variables $x_1, \dots, x_n$ and $\phi(t_1, \dots, t_n)$ denote the formula obtained by replacing each free occurrence of x_i in ϕ by the term t_i.

$(\neg\forall x.\psi(x)) \to \exists x.\neg\psi(x)$ is a theorem

$\vdash (\neg\forall x.\psi(x)) \to \exists x.\neg\psi(x)$ — $\to_\mathsf{I}$
- **1.** $\neg\forall x.\psi(x) \vdash \exists x.\neg\psi(x)$ — RAA
  - **1.** $\neg\forall x.\psi(x), \neg\exists x.\neg\psi(x) \vdash \perp$ — $\neg_\mathsf{E}$
    - **1.** $\neg\forall x.\psi(x), \neg\exists x.\neg\psi(x) \vdash \neg\forall x.\psi(x)$ — assumption
    - **2.** $\neg\forall x.\psi(x), \neg\exists x.\neg\psi(x) \vdash \forall x.\psi(x)$ — $\forall_\mathsf{I}$
      - **1.** $\neg\forall x.\psi(x), \neg\exists x.\neg\psi(x) \vdash \psi(x_0)$ — RAA
        - **1.** $\neg\forall x.\psi(x), \neg\exists x.\neg\psi(x), \neg\psi(x_0) \vdash \perp$ — $\neg_\mathsf{E}$
          - **1.** $\neg\forall x.\psi(x), \neg\exists x.\neg\psi(x), \neg\psi(x_0) \vdash \neg\exists x.\neg\psi(x)$ — assumption
          - **2.** $\neg\forall x.\psi(x), \neg\exists x.\neg\psi(x), \neg\psi(x_0) \vdash \exists x.\neg\psi(x)$ — $\exists_\mathsf{I}$
            - **1.** $\neg\forall x.\psi(x), \neg\exists x.\neg\psi(x), \neg\psi(x_0) \vdash \neg\psi(x_0)$ — assumption

Soundness and completeness of $\mathcal{N}_{\mathsf{FOL}}$

Soundness

If $\Gamma \vdash \phi$, then $\Gamma \models \phi$.

Completeness

If $\Gamma \models \phi$, then $\Gamma \vdash \phi$.

Deductive approach vs semantic approach

- Deductive approach
  - Based on a proof system.
  - The goal is to prove that a formula is valid.
  - The tools based on this approach are called proof-assistants and allow the interactive development of proofs.
  - In the proof process a derivation (proof tree) is constructed.
- Semantic approach
  - Based on the notion of model.
  - The goal is to prove that a set of formulas is satisfiable.
  - The SMT-solvers are tools based on this approach, which are decision procedures that produce a "SAT/UNSAT/UNKNOW" answer.
  - If the answer is SAT, a model is produced.

Higher-order logic

There is no need to stop at first-order logic; one can keep going.

- We can add to the language "super-predicate" symbols, which take as arguments both individual symbols and predicate symbols. And then we can allow quantification over super-predicate symbols.
- And we can keep going further...
- We reach the level of *type theory*.

Higher-order logics allows quantification over "everything".

- One needs to introduce some kind of typing scheme.
- The original motivation of Church (1940) to introduce simple type theory was to define higher-order (predicate) logic.

Proposition as Types

Two branches of formal logic: *classical* and *intuitionistic*

- The classical understanding of logic is based on the notion of **truth**. The truth of a statement is "absolute" and independent of any reasoning, understanding, or action. So, statements are either true or false, and $(A \vee \neg A)$ must hold no matter what the meaning of A is.
- Intuitionistic (or constructive) logic is a branch of formal logic that rejects this guiding principle. It is based on the notion os **proof**. The judgement about a statement is based on the existence of a proof (or "construction") of that statement.

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, $\neg\neg\phi \to \phi$ (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 $\phi \to \psi$ 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 $\phi(d)$.

Intuitionistic logic

Some classical tautologies that are not intuitionistically valid

$\phi \vee \neg\phi$	*excluded middle law*
$\neg\neg\phi \to \phi$	*double negation law*
$((\phi \to \psi) \to \phi) \to \phi$	*Pierce's law*
$(\phi \to \psi) \vee (\psi \to \phi)$	
$(\phi \to \psi) \to (\neg\phi \vee \psi)$	
$\neg(\phi \wedge \psi) \to (\neg\phi \vee \neg\psi)$	
$(\neg\phi \to \psi) \to (\neg\psi \to \phi)$	
$(\neg\phi \to \neg\psi) \to (\psi \to \phi)$	
$\neg\forall x.\, \neg\phi(x) \to \exists x.\, \phi(x)$	
$\neg\exists x.\, \neg\phi(x) \to \forall x.\, \phi(x)$	
$\neg\forall x.\, \phi(x) \to \exists x.\, \neg\phi(x)$	

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 *reductio ad absurdum* rule (RAA).

- Traditionally, classical logic is defined by extending intuitionistic logic with the *reductio ad absurdum* law, 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 $\lambda\!\to$

Implicational fragment of PL	$\lambda\!\to$
$\dfrac{\phi \in \Gamma}{\Gamma \vdash \phi}$ (assumption)	$\dfrac{(x:\phi) \in \Gamma}{\Gamma \vdash x:\phi}$ (var)
$\dfrac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \to \psi}\ (\to_I)$	$\dfrac{\Gamma, x:\phi \vdash e:\psi}{\Gamma \vdash (\lambda x{:}\phi.e) : \phi \to \psi}$ (abs)
$\dfrac{\Gamma \vdash \phi \to \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi}\ (\to_E)$	$\dfrac{\Gamma \vdash a : \phi \to \psi \qquad \Gamma \vdash b : \phi}{\Gamma \vdash (a\ b) : \psi}$ (app)

The Curry-Howard isomorphism

The *proposition-as-types* interpretation 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
- *proof checking* boils down to *type checking*.

This analogy between systems of formal logic and computational calculi was first discovered by Haskell Curry and William Howard.

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.

Type-theoretic approach to interactive theorem proving

provability of formula A	$\Longleftrightarrow$	inhabitation of type A
proof checking	$\Longleftrightarrow$	type checking
interactive theorem proving	$\Longleftrightarrow$	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.

Higher-order logic and type theory

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

$$A, B, M, N \ ::= \ s \mid x \mid M\,N \mid \lambda x\!:\!A.M \mid \Pi x\!:\!A.\,B$$

$x \in \mathcal{V}$ (a countable set of *variables*) and $s \in \mathcal{S}$ (a set of *sorts*).

- Both Π and λ bind variables.
- Both $\Rightarrow$ and $\forall$ are generalized by a single construction Π.
  We write $A \to B$ instead of $\Pi x\!:\!A.\,B$ whenever $x \notin \mathrm{FV}(B)$.
- The typing rules for abstraction and application became

$$\text{(abs)} \quad \frac{\Gamma, x\!:\!A \vdash M : B \qquad \Gamma \vdash (\Pi x\!:\!A.\,B) : s}{\Gamma \vdash (\lambda x\!:\!A.M) : (\Pi x\!:\!A.\,B)}$$

$$\text{(app)} \quad \frac{\Gamma \vdash M : (\Pi x\!:\!A.\,B) \qquad \Gamma \vdash N : A}{\Gamma \vdash M N : B[N/x]}$$

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 (1967).
- Modern proof assistants, aggregate to the proof checker a proof-development system for helping the user to develop the proofs interactively.
- In a proof-assistant, after formalizing the primitive notions of the theory (under study), the user develops the proofs interactively by means of (proof) tactics, and when a proof is finished a "proof term" (or simply a "proof script") is created.

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.
  - 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), **Isabelle** (based on intuitionistic simple type theory).
- 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).

The reliability of machine checked proofs

- Machine assisted theorem proving:
  - helps to deal with large problems;
  - prevents us from overseeing details;
  - does the bookkeeping of the proofs.

- **But, 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.

Coq in Brief

The Coq proof assistant

- The Coq system is a formal proof management system that
  - allows the expression of mathematical assertions, and mechanically checks proofs of these assertions;
  - helps to find formal proofs;
  - extracts a certified program from the constructive proof of its formal specification.

- Typical applications include the formalization of mathematics and the formalization of programming languages semantics.

- The underlying formal language of Coq is a *Calculus of Constructions* with *inductive definitions*:

  the Calculus of Inductive Constructions (CIC)

The Coq proof assistant

Main features:

- interactive theorem proving
- functional programming language
- powerful specification language
  (includes dependent types and inductive definitions)
- tactic language to build proofs
- type-checking algorithm to check proofs

More concrete stuff:

- 3 sorts to classify types: Prop, Set, Type
- inductive definitions are primitive
- elimination mechanisms on such definitions

The Coq proof assistant

CIC all objects have a type. There are

- types for functions (or programs)
- atomic types (especially datatypes)
- types for proofs
- types for the types themselves.

Types are classified by the three basic sorts

- Prop *(logical propositions)*
- Set *(mathematical collections)*
- Type *(abstract types)*

which are themselves atomic abstract types.

Coq syntax

$\lambda x\!:\!A.\, \lambda y\!:\!A \to B.\, y\, x$ `fun (x:A) (y:A->B) => y x`

$\forall x\!:\!A.\, P(x) \to P(x)$ `forall x:A, P x -> P x`

Inductive types

```
Inductive nat :Set  :=  O : nat
                     |  S : nat -> nat.
```

This definition yields: – constructors: `O` and `S`
– recursors: `nat_ind`, `nat_rec` and `nat_rect`

General recursion and case analysis

```
Fixpoint double (n:nat) :nat :=
   match n with
     | O => O
     | (S x) => S (S (double x))
end.
```


Environment

In the Coq system the well typing of a term depends on an environment which consists in a *global environment* and a *local context*.

- The **local context** is a sequence of variable declarations, written $x : A$ (A is a type) and "standard" definitions, written $x := t : A$ (that is abbreviations for well-formed terms).
- The **global environment** is the list of global declarations and definitions. This includes not only assumptions and "standard" definitions, but also definitions of inductive objects. (The global environment can be set by loading some libraries.)

We frequently use the names *constant* to describe a globally defined identifier and *global variable* for a globally declared identifier.

The typing judgments are as follows:

$$E \,|\, \Gamma \vdash t : A$$

Declarations and definitions

The environment combines the contents of initial environment, the loaded libraries, and all the global definitions and declarations made by the user.

Loading modules

`Require Import ZArith.`

This command loads the definitions and declarations of module `ZArith` which is the standard library for basic relative integer arithmetic.

The Coq system has a block mechanism (similar to the one found in many programming languages) `Section` *id.* ... `End` *id.* which allows to manipulate the local context (by expanding and contracting it).

Declarations

```
Parameter max_int :  Z.               Global variable declaration
Section Example.
Variables A B : Set.                  Local variable declarations
Variables Q : Prop.
Variables (b:B) (P : A->Prop).
```

Declarations and definitions

Definitions

```
Definition min_int := (1 - max_int)
```
Global definition

```
Let FB := B -> B.
```
Local definition

Proof-terms

```
Lemma trivial :  forall x:A, P x -> P x.
intros x H.
exact H.
Qed.
```

- Using tactics a term of type `forall x:A, P x -> P x` has been created.
- Using the Qed command the identifier `trivial` is defined as this proof-term and add to the global environment.

Computation

Computations are performed as series of *reductions*. The `Eval` command computes the normal form of a term with respect to some reduction rules (and using some reduction strategy: `cbv` or `lazy`).

β-reduction for compute the value of a function for an argument:

$$(\lambda x : A.\, a)\, b \quad \to_\beta \quad a[b/x]$$

δ-reduction for unfolding definitions:

$$e \quad \to_\delta \quad t \qquad \text{if } (e := t) \in E \,|\, \Gamma$$

ι-reduction for primitive recursion rules, general recursion, and case analysis

ζ-reduction for local definitions: $\texttt{let } x := a \texttt{ in } b \quad \to_\zeta \quad b[a/x]$

Note that the conversion rule is

$$\frac{E \,|\, \Gamma \vdash t : A \qquad E \,|\, \Gamma \vdash B : s}{E \,|\, \Gamma \vdash t : B} \quad \text{if } A =_{\beta\iota\delta\zeta} B \text{ and } s \in \{\mathsf{Prop}, \mathsf{Set}, \mathsf{Type}\}$$

Proof example

```
Section EX.

Variables (A:Set) (P : A->Prop).
Variable Q:Prop.

Lemma example : forall x:A, (Q -> Q -> P x) -> Q -> P x.
Proof.
intros x h g.
apply h.
assumption.
assumption.
Qed.
```

$\texttt{example} = \lambda x{:}A.\lambda h{:}Q \to Q \to P\,x.\lambda g{:}Q.\, h\, g\, g$

```
Print example.
example =
fun (x : A) (h : Q -> Q -> P x) (g : Q) => h g g
     : forall x : A, (Q -> Q -> P x) -> Q -> P x
```


Proof example

Observe the analogy with the lambda calculus.

$\texttt{example} = \lambda x{:}A.\lambda h{:}Q \to Q \to P x.\lambda g{:}Q.\, h\, g\, g$

$A : \mathsf{Set}, P : A \to \mathsf{Prop}, Q : \mathsf{Prop} \vdash \texttt{example} : \forall x{:}A, (Q \Rightarrow Q \Rightarrow Px) \Rightarrow Q \Rightarrow Px$

```
End EX.
Print example.
example =
fun (A:Set) (P:A->Prop) (Q:Prop) (x:A) (h:Q->Q->P x) (g:Q) => h g g
     : forall (A : Set) (P : A -> Prop) (Q : Prop) (x : A),
       (Q -> Q -> P x) -> Q -> P x
```

$\vdash \texttt{example} : \forall A{:}\mathsf{Set}, \forall P{:}A \to \mathsf{Prop}, \forall Q{:}\mathsf{Prop}, \forall x{:}A, (Q \Rightarrow Q \Rightarrow P\,x) \Rightarrow Q \Rightarrow P\,x$

Tactics for first-order reasoning

Proposition (P)	Introduction	Elimination (H of type P)
$\bot$		`elim` H, contradiction
$\neg A$	`intro`	`apply` H
$A \wedge B$	`split`	`elim` H, `destruct` H `as [H1 H2]`
$A \Rightarrow B$	`intro`	`apply` H
$A \vee B$	`left`, `right`	`elim` H, `destruct` H `as [H1|H2]`
$\forall x : A.\, Q$	`intro`	`apply` H
$\exists x : A.\, Q$	`exists` *witness*	`elim` H, `destruct` H `as [x H1]`

Some more tactics

Some basic tactics

- `intro`, `intros` – introduction rule for Π (several times)
- `apply` – elimination rule for Π
- `assumption` – match conclusion with an hypothesis
- `exact` – gives directly the exact proof term of the goal

Some automatic tactics

- `trivial` – tries those tactics that can solve the goal in one step.
- `auto` – tries a combination of tactics `intro`, `apply` and `assumption` using the theorems stored in a database as hints for this tactic.
- `tauto` – useful to prove facts that are tautologies in intuitionistic PL.
- `intuition` – useful to prove facts that are tautologies in intuitionistic PL.
- `firstorder` – useful to prove facts that are tautologies in intuitionistic FOL.

Coq - software, documentation, contributions, tutorials

http://coq.inria.fr/

Exercises

Load the file `lesson1.v` in the Coq proof assistant. Analyse the examples and solve the exercises proposed.