2.7.1 — Foundations of Proof Systems

2017-2018

Chapter 1

First Order Logic

We recall in this section some definitions about *first order logic*. We assume, throughout the section, a countably infinite set of variables $\mathcal{X} = \{x, y, z, ...\}$, a ranked[1] set $\Sigma = \{f, g, h, ...\}$ of *function symbols* and a ranked set $\mathcal{P} = \{P, Q, R, ...\}$ of *predicates*.

Definition 1.0.1 (Terms, formulas)**.** The set of *term* $\mathcal{T}$ is inductively defined as follows:

$$\begin{array}{rll} t, u, v \in \mathcal{T} := & x & (x \in \mathcal{X}) \\ & \mid\ f(t_1, \dots, t_n) & (f_{/n} \in \Sigma) \end{array}$$

The set of formulas $\mathcal{F} = \{\phi, \psi, ...\}$ is inductively defined as follows:

$$\begin{array}{rll} \phi, \psi \in \mathcal{F} := & P(t_1, \cdots, t_n) & (P_{/n} \in \mathcal{P}) \\ & \mid\ \neg\phi & (\text{negation}) \\ & \mid\ \phi \wedge \psi & (\text{conjunction}) \\ & \mid\ \phi \vee \psi & (\text{disjunction}) \\ & \mid\ \phi \implies \psi & (\text{implication}) \\ & \mid\ \forall x.\, \phi & (\text{universal quantification}) \\ & \mid\ \exists x.\, \phi & (\text{existential quantification}) \end{array}$$

We use the usual mathematical conventions for the precedence and associativity of the logical connectors. We write $\phi \iff \psi$ for $(\phi \implies \psi) \wedge (\psi \implies \phi)$. We write $\mathrm{FV}(t)$ (resp. $t[x \mapsto u]$) for the set of variables that appear in t (for the formal substitution of the variable x by the term u in t). Likewise, we write $\mathrm{FV}(\phi)$ (resp. $\phi[x \mapsto u]$) for the set of *free variables* of the formula ϕ (resp. for the capture-free substitution of the variable x by the term u in ϕ). A term t is ground if $\mathrm{FV}(t) = \emptyset$. Likewise, a formula ϕ is *closed* if $\mathrm{FV}(\phi) = \emptyset$.

We now recall the notion of a valid formula w.r.t. a first-order structure (or model, or interpretation):

Definition 1.0.2 (First-order structure)**.** A *first-order structure* is a triple $\mathcal{M} = (\mathcal{D}, \iota, \rho)$ where $\mathcal{D}$ is a non-empty set (the *domain of discourse*), ι is a map associating to $f_{/n} \in \Sigma$ a function $\iota_f : \mathcal{D}^n \to \mathcal{D}$ and ρ is a map associating to $P_{/n} \in \mathcal{P}$ a n-ary relation $\rho_P \in \mathcal{D}^n$.

[1] A ranked set A if a set whose elements are equipped with an arity (i.e. a natural number) — we write $a_{/n} \in A$ if $a \in A$ with associated arity n.

Definition 1.0.3 (Interpretation of a term, of a formula)**.** Let $\mathcal{M} = (\mathcal{D}, \iota, \rho)$ be a first-order structure and let $\nu : \mathcal{X} \to \mathcal{D}$ be a *valuation*. The *evaluation* of a term t w.r.t. $\mathcal{M}$ and ν, written $[\![t]\!]_\nu^{\mathcal{M}}$ ($\in \mathcal{D}$) or $[\![t]\!]_\nu$ when $\mathcal{M}$ is clear from the context, is inductively defined as:

$$[\![x]\!]_\nu = \nu(x) \qquad\qquad [\![f(t_1, \dots, t_n)]\!] = \iota_f([\![t_1]\!]_\nu, \dots, [\![t_n]\!]_\nu).$$

The *evaluation* of a formula ϕ w.r.t. $\mathcal{M}$ and ν, written $[\![\phi]\!]_\nu^{\mathcal{M}}$ ($\in \{\top, \bot\}$) or $[\![\phi]\!]_\nu$ when $\mathcal{M}$ is clear from the context, is inductively defined as:

$$\begin{aligned}
[\![P(t_1, \dots, t_n)]\!] &= \top \text{ if } \rho_P([\![t_1]\!]_\nu, \dots, [\![t_n]\!]_\nu) \\
[\![\neg\phi]\!] &= \top \text{ if } [\![\phi]\!] = \bot \\
[\![\phi \wedge \psi]\!] &= \top \text{ if } [\![\phi]\!] = \top \text{ and } [\![\psi]\!] = \top \\
[\![\phi \vee \psi]\!] &= \top \text{ if } [\![\phi]\!] = \top \text{ or } [\![\psi]\!] = \top \\
[\![\phi \implies \psi]\!] &= \top \text{ if } [\![\phi]\!] = \top \text{ implies } [\![\psi]\!] = \top \\
[\![\forall x.\, \phi]\!] &= \top \text{ if for any } v \in \mathcal{D},\ [\![\phi]\!]_{\nu[x \mapsto v]} = \top \\
[\![\forall x.\, \phi]\!] &= \top \text{ if there exists } v \in \mathcal{D},\ [\![\phi]\!]_{\nu[x \mapsto v]} = \top \\
[\![\phi]\!] &= \bot \text{ otherwise.}
\end{aligned}$$

A formula ϕ *holds* in a model $\mathcal{M}$ and for a valuation ν, written $\mathcal{M}, \nu \vDash \phi$, iff $[\![\phi]\!]_\nu^{\mathcal{M}} = \top$. A formula ϕ is *satisfiable in a model* $\mathcal{M}$ (or the *model* $\mathcal{M}$ *satisfies* ϕ) iff there exists a valuation ν s.t. $\mathcal{M}, \nu \vDash \phi$. It is *valid in a model* $\mathcal{M}$, written $\mathcal{M} \vDash \phi$, iff $\mathcal{M}, \nu \vDash \phi$ for any valuation ν. Last, a formula ϕ is *valid*, written $\vDash \phi$, iff $\mathcal{M} \vDash \phi$ for any model $\mathcal{M}$.

We conclude this section with the definition of a first-order theory:

Definition 1.0.4 (First-order theory)**.** A *first-order theory* w.r.t. a signature Σ and a set of predicated $\mathcal{P}$ is defined by a set of *closed* formulas called *axioms* of the theory. A formula ϕ is valid in a theory $\mathcal{T}$, written $\mathcal{T} \vDash \phi$, iff for any model $\mathcal{M}$ that satisfies the axioms of $\mathcal{T}$, we have $\mathcal{M} \vDash \phi$. A theory $\mathcal{T}$ is said to be *consistent* iff model $\mathcal{M}$ that satisfies all the axioms but not $\bot$ — i.e. if $\bot$ is not valid in $\mathcal{T}$.

1.1 Natural Deduction

Natural deduction is a formal system for first order logic that has been introduced by Gerhard Gentzen in 1934. In contrary to the Hilbert's systems that are based on a set of axioms, natural deduction relies on a set of dual introduction/elimination rules for each connector of the logic.

Definition 1.1.1 (Natural Deduction Rules)**.** We call *sequent* any pair $\Gamma \vdash \phi$, where Γ is a finite set of formulas and ϕ is a formula. A sequent $\Gamma \vdash \phi$ is *provable (in natural deduction)* if it can be inductively derived from the rules of Figure 1.1. A *proof (in natural deduction)* of a sequent $\Gamma \vdash \phi$ is a tree:

- where nodes are labelled by a sequent,
- whose root is labelled by the sequent $\Gamma \vdash \phi$, and

NON STRUCTURAL RULES

$$\frac{\phi \in \Gamma}{\Gamma \vdash \phi}\ (\mathsf{Ax}) \qquad \frac{}{\Gamma \vdash \phi \vee \neg\phi}\ (\mathsf{EM})$$

INTRODUCTION RULES

$$\frac{\Gamma \vdash \phi \qquad \Gamma \vdash \neg\phi}{\Gamma \vdash \bot}\ (\bot\text{-I}) \qquad \frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi}\ (\vee\text{-I}_1) \qquad \frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi}\ (\vee\text{-I}_1)$$

$$\frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi}\ (\vee\text{-I}_2) \qquad \frac{\Gamma \vdash \phi \qquad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi}\ (\wedge\text{-I}) \qquad \frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \implies \psi}\ (\Rightarrow\text{-I})$$

$$\frac{\Gamma \vdash \phi \qquad x \notin \mathrm{FV}(\Gamma)}{\Gamma \vdash \forall x.\, \phi}\ (\forall\text{-I}) \qquad \frac{\Gamma \vdash \phi[x \mapsto t]}{\Gamma \vdash \exists x.\, \phi}\ (\exists\text{-I})$$

ELIMINIATION RULES

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \phi}\ (\bot\text{-E}) \qquad \frac{\Gamma \vdash \phi \vee \psi \qquad \Gamma, \phi \vdash \eta \qquad \Gamma, \psi \vdash \eta}{\Gamma \vdash \eta}\ (\vee\text{-E}) \qquad \frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \phi}\ (\wedge\text{-E}_1)$$

$$\frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \psi}\ (\wedge\text{-E}_2) \qquad \frac{\Gamma \vdash \phi \implies \psi \qquad \Gamma \vdash \phi}{\Gamma \vdash \psi}\ (\Rightarrow\text{-E}) \qquad \frac{\Gamma \vdash \forall x.\, \phi}{\Gamma \vdash \phi[x \mapsto t]}\ (\forall\text{-E})$$

$$\frac{\Gamma \vdash \exists x.\, \phi \qquad \Gamma, \phi \vdash \psi \qquad x \notin \mathrm{FV}(\Gamma, \psi)}{\Gamma \vdash \psi}\ (\exists\text{-E})$$

Figure 1.1: Natural Deduction Rules

- s.t. for any node labelled with the sequent $\Delta \vdash \psi$ and whose children are resp. labelled with sequents $\Delta_i \vdash \psi_i$ ($i \in \{1..n\}$), there exists a rule of Figure 1.1 of the form:

$$\frac{\Delta_1 \vdash \psi_1 \quad \cdots \quad \Delta_n \vdash \psi_n}{\Delta \vdash \psi}$$

In that case, we say that the sequent $\Gamma \vdash \phi$ is *provable (in natural deduction)*. A proposition ϕ is provable if the sequent $\vdash \phi$ is provable.

Example. Figure 1.2 gives a proof in natural deduction of the proposition $\phi \wedge \psi \implies \psi \wedge \phi$.

Definition 1.1.2 (Natural deduction proof in a theory)**.** Let $\mathcal{T}$ be a first-order theory. A formula ϕ is provable in $\mathcal{T}$, written $\mathcal{T} \vdash \phi$, iff there exists a finite subset Γ of the axioms of $\mathcal{T}$ s.t. $\Gamma \vdash \phi$.

Definition 1.1.3 (Intuitionistic proof)**.** A proof is said to be *intuitionistic*, or done in *intuitionistic logic*, if it does not use the rule of excluded-middle (EM). A proposition is constructively provable if there exists a constructive proof for it.

$$\dfrac{\dfrac{\dfrac{}{\phi \wedge \psi \vdash \phi \wedge \psi}\,(\text{Ax})}{\phi \wedge \psi \vdash \psi}\,(\wedge\text{-E}_2) \qquad \dfrac{\dfrac{}{\phi \wedge \psi \vdash \phi \wedge \psi}\,(\text{Ax})}{\phi \wedge \psi \vdash \phi}\,(\wedge\text{-E}_1)}{\phi \wedge \psi \vdash \psi \wedge \phi}\,(\wedge\text{-I})}{\vdash \phi \wedge \psi \implies \psi \wedge \phi}\,(\Rightarrow\text{-I})$$

Figure 1.2: Proof in Natural Deduction of $\phi \wedge \psi \implies \psi \wedge \phi$

Definition 1.1.4 (Minimal logic)**.** A proof is said to be done in *minimal logic* if it uses neither the excluded middle rule, nor the rule of falsehood elimination ($\bot$-E).

Key properties of natural deduction are its correctness and completeness. w.r.t. the notation of validity:

Lemma 1.1.1 (Correctness)**.** *If $\Gamma \vdash \phi$, then $\vDash \bigwedge_{\psi \in \Gamma} \psi \implies \phi$. Likewise, if $\mathcal{T} \vdash \phi$, then $\mathcal{T} \vDash \phi$.*

Lemma 1.1.2 (Completeness)**.** *If $\vDash \bigwedge_{\psi \in \Gamma} \psi \implies \phi$, then $\Gamma \vdash \phi$. Likewise, if $\mathcal{T} \vDash \phi$, then there exists a finite subset Γ of axioms of $\mathcal{T}$ s.t. $\Gamma \vdash \phi$.*

1.2 Theories

First-order logic is the most commonly used logical formalism, and *regular* mathematics are supposed to be done in set theory. Set theory is one of many possible *theories* of first-order logic. As we have seen above, a theory is defined in two steps: i) the language which is given by a first-order structure, and ii) the truth in the theory is defined by the combination of the deduction rules with the axioms of the theory.

1.2.1 Arithmetic

Arithmetic is a remarkable theory for various reasons. For instance, it was defined as early as 1889 by Guiseppe Peano (see [3]). But also, it is the simplest "complex" theory, in the sense that it is undecidable, and that it verifies Gödel's incompleteness theorem. The objects of arithmetic are essentially the natural numbers built from the following symbols:

- 0 of arity 0,
- S, the successor function, of arity 1, and
- $+$ and $\times$, the addition and multiplication, of arity 2.

Both $+$ and $\times$ are used in infix form, using the usual syntax of mathematics.

Arithmetic has only one predicate symbol: equality. It is of arity 2, usually written $=$ in infix notation.

The axioms of arithmetic are:

$$\forall x.\, 0 + x = x \tag{1}$$
$$\forall x\, y.\, S(x) + y = S(x + y) \tag{2}$$

$$\forall x.\, 0 \times x = 0 \tag{3}$$
$$\forall x\, y.\, S(x) \times y = y + x \times y \tag{4}$$

$$\forall x.\, \neg(0 = S(x)) \tag{5}$$
$$\forall x\, y.\, S(x) = S(y) \implies x = y \tag{6}$$

$$[P]\ \ P(0) \land (\forall x.\, P(x) \implies P(S(x))) \implies \forall x.\, P(x). \tag{7}$$

We see that the first four axioms describe the addition and multiplication operations. It is also important to note that the last axiom, induction, is actually an axiom scheme: there is one axiom for each property P. Another, more precise way to describe it is to say that for every proposition P well-formed in the language of arithmetic, and every variable x, the following axiom holds:

$$P[x \mapsto 0] \land (\forall x.\, P \implies P[x \mapsto S(x)]) \implies \forall x.\, P.$$

Finally, we need axioms describing equality. We can use a slight variation with respect to Peano's original presentation by taking one axiom (reflexivity) and one scheme:

$$\forall x.\, x = x \tag{8}$$
$$[P]\ \ \forall x.\, \forall y.\, P(x) \land x = y \implies P(y). \tag{9}$$

The last scheme corresponds to what is often called *Leibniz' equality*, because Leibniz had described equality by stating that two objects are equal when they verify exactly the same set of properties.

1.2.2 Set Theory

Set theory is a very powerful formalism but it is not the best suited for use in a proof system. We therefore do not study it in depth in this course, but give a quick overview.

The language of set theory is very minimal. There are no function symbols, and only two predicate symbols: membership, written $\in$, and equality $=$. Both are binary predicates written using infix notation.

The axioms of what is called Zermelo's set theory are:

Extensionality — two sets are equal if and only if they have the same elements:

$$\forall a\, b.\, (\forall c.\, c \in a \iff c \in b) \implies a = b$$

Pair — for any pair of sets a and b, there exists a set, written $\{a, b\}$, whose elements are exactly a and b:

$$\forall a\, b.\, \exists c.\, (\forall d.\, d \in c \iff d = a \vee d = b)$$

On can replace this formulation of the pair axiom by adding a binary function symbol Pair together with the axiom

$$\forall a\, b\, d.\, d \in \mathsf{Pair}(a, b) \iff d = a \vee d = b.$$

Elementary sets — there exists a set:

$$\exists x.\, x = x$$

This axiom can be replaced by: *there exists a set that does not contain any elements* — $\exists a.\, \forall x.\, \neg(x \in a)$. By extentionality, this set is necessarily unique. It is called the *empty set* and is written $\emptyset$.

One can replace this formulation of the axiom by adding a constant symbol $\emptyset$ and the axiom $\forall x.\, x \notin \emptyset$.

Union — for any fixed set a, there exists the set of the elements of elements of a:

$$\forall a.\, \exists b.\, (\forall c.\, c \in b \iff \exists d.\, d \in a \wedge c \in d)$$

One can replace this formulation by adding a unary function symbol Union togather with the axiom $\forall a\, b.\, b \in \mathsf{Union}(a) \iff \exists d.\, b \in d \wedge d \in a$.

Comprehension scheme — for every set a and property P, there exists a set, written $\{x \in a \mid P(x)\}$, whose elements are exactly the elements of a that validate P:

$$\forall a.\, \exists b.\, (\forall c.\, c \in b \iff c \in a \wedge P(c))$$

Powerset — for any fixed set a, there exists a set whose elements are exactly the subsets of a:

$$\forall a.\, \exists b.\, (\forall c.\, c \in b \iff c \subseteq a)$$

where $a \subseteq b$ is a notation for the predicate $\forall x.\, x \in a \implies x \in b$.

This set of axioms can be extended to give the Zermelo-Fraenkel set theory (ZF) and the axiom of choice (giving ZFC). The study of set theory is a vast subject which goes beyond the scope of this course. An excellent french reference is [2].

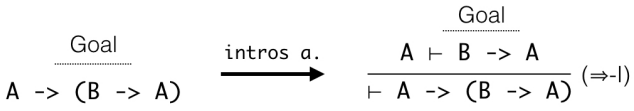
1.3 Formal proof construction

The formalism implemented by Coq includes first-order logic. It includes actually much more, but in particular a statement *and* a proof in first-order logic can straightforwardly be translated to Coq. Furthermore, the translated proof follows the same tree-structure as the original natural deduction FOL proof.

The generic way to construct a proof is top-down (although this terminology is somewhat misleading in the case of natural deduction because the conclusion is written at the bottom). On starts by stating the statement to be proved. Say:

```
Lemma ex1 : A -> (B -> A).
```

The proof is constructed through commands called proof tactics. The most basic tactics can directly be related to the natural deduction rules. In the case above, we can apply the tactic corresponding to the $\Longrightarrow$ -I rule: `intro a`. It will create a new goal `B->A` with a new hypothesis `A`. In other words, we will have constructed a new *partial proof tree* of the following form:



In general, active goals are leafs in an incomplete proof term. Some tactics can create more than one new subgoal, like `split` which corresponds to the introduction of the conjunction.

We do not detail the Coq tactics here.

When is the proof checked?

The aim of the mechanical proof checking is to achieve a very high degree of certainty that the proof is actually correct. This is related, at the same time, to the formalism and to the software architecture of the proof system. This question is more interesting than one may think at first sight. In the case of Coq, once the proof is completed, the proof-tree is checked and then stored through the command `Qed`. The part of the Coq software which is critical for the trust to the system is the routine performing this last check. We may thus note that it relies on the fact that checking the correctness of a proof derivation is decidable. This point is obvious for first-order logic, what will become more delicate when the formalism will get more complex as we advance through the course.

1.4 Deduction modulo

A first step in making the formalism more complex is deduction modulo.

Deduction modulo is an generalization of first-order logic, in which the language is equipped with a congruent equivalence relation $=_R$, typically induced by a rewriting relation.

Logically, one the identifies propositions modulo $=_R$, which means we add the following deduction rule:

$$(\textsc{Conv})\frac{\Gamma \vdash \phi}{\Gamma \vdash \psi} \text{ if } \phi =_R \psi$$

In general we want the relation $=_R$ to be decidable in order to ba able to check the validity of a proof-tree. Typically, it will be terminating and confluent.

1.4.1 Arithmetic: simple rewrites

A useful feature of deduction modulo is that it allows to replace some deduction steps by computations. Let us keep the language of arithmetic and add the following rewrite rules:

$$\begin{aligned} 0 + x &\rhd x \\ S(x) + y &\rhd S(x + y) \\ O \times x &\rhd 0 \\ S(x) \times y &\rhd y + x \times y \end{aligned}$$

Once we have done this, we can drop the first four axioms of arithmetic which have become redundant. Furthermore, some proofs become shorter in this new presentation of arithmetic. For instance, the proof of 2+2=4 boils down to one single deduction step:

$$\dfrac{\dfrac{\dfrac{}{\forall x, x = x}\ (\text{Ax})}{S(S(S(S(0)))) = S(S(S(S(0))))}\ (\forall\text{-E})}{S(S(0)) + S(S(0)) = S(S(S(S(0))))}\ (\text{Conv})$$

Since many computation steps can be packed in a single use of the conversion rule, the proof trees can be arbitrarily smaller than in the conventional presentation of arithmetic in which one needs to use one more inference for every computation step.

1.4.2 Arithmetic: more complex rewrite rules

One sees above that, in deduction modulo, *rewrite rules replace axioms*. While this is straightforward for the first four axiom of arithmetic, it is also possible for the others properties:

- Adding a predecessor function pred with the rewrite rule

  $$\text{pred}(S(x)) \rhd x$$

  allows to drop the axiom

  $$\forall x\, y, S(x) = S(y) \Rightarrow x = y.$$

- We can prove the property $\forall x, 0 \neq S(x)$ by adding a new predicate symbol D and two rewrite rules:

  $$\begin{aligned} D(0) &\rhd \bot \\ D(S(x)) &\rhd \top \end{aligned}$$

In both case, we leave it as an exercise to check that the rewrite rule(s) allow(s) to prove the corresponding axiom. This can be performed in Coq.

1.5 Relations with the formalism of Coq

Coq implements a complex type theory which will be better described later. For now, let us mention some facts:

- Peano's arithmetic is a fragment of Coq's type theory.
- Coq's logic has a conversion rule. Not every rewrite system can be implemented in Coq. But the rewrite rules given above in the context of arithmetic actually are considered by Coq.

One can check reductions by commands like:

```
Eval compute in (2 + 2).
```

which will give out 4 as a result. This corresponds to the given rewrite rules, considering that 2 (resp. 4) is just pretty-printing for $(S\ (S\ 0))$ (resp. $(S\ (S\ (S\ (S\ 0))))$).

Cuts, cut elimination & cut-free proofs

The questions of cuts and cut-elimination are central in proof theory. In this year's course we chose to post-pone its study after the Curry-Howard isomorphism.

1.6 Logical cuts

Roughly, a cut in a proof can be understood as a way to prove a general statement only once. For instance when proving $(2+x)^2 = x^2 + 4x + 4$ we can

- either use the result $\forall a\, b, (a+b)^2 = a^2 + b^2 + 2ab$ and instantiate a and b by x and 2,
- or do a proof from scratch, which will have the same structure as the generic one, but only considers 2 and x.

The first option is a proof with a *cut*. In practice, being able to use such cuts is essential for making mathematics tractable. In theory however, cuts are redundant and can be eliminated. The corresponding cut-elimination theorems are important for various results like consistency or the completeness of automated deduction procedures.

To make things simple, we can say that:

- Proofs with cuts can be shorter, because some (parts of the) proof(s) can be reused and shared.
- But proofs without cuts have some interesting structural properties.

Definition 1.6.1 (Cut in Natural Deduction)**.** In the context of natural deduction, a *logical cut* is a proof that contains an elimination rule whose first premise in a introduction rule of the same connector. Figure 1.3 gives an extensive listing of possible cuts. A proof that does not contain any cut is said to be *cut-free*.

1.7 Properties of cut-free proofs

Lemma 1.7.1. *A cut-free proof of* $[] \vdash A$ *ends with an introduction rule.*

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \psi} \qquad \dfrac{\vdots}{\Gamma \vdash \neg\psi}}{\dfrac{\Gamma \vdash \bot}{\Gamma \vdash \phi}\ (\bot\text{-E})}\ (\bot\text{-I}) \qquad \dfrac{\dfrac{\dfrac{\vdots}{\Gamma, \phi \vdash \psi}}{\Gamma \vdash \phi \implies \psi}\ (\Rightarrow\text{-I}) \qquad \dfrac{\vdots}{\Gamma, \phi \vdash \psi}}{\Gamma \vdash \psi}\ (\Rightarrow\text{-E})$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \phi}}{\Gamma \vdash \phi \vee \psi}\ (\vee\text{-I}_1) \qquad \dfrac{\vdots}{\Gamma, \phi \vdash \eta} \qquad \dfrac{\vdots}{\Gamma, \psi \vdash \eta}}{\Gamma \vdash \eta}\ (\vee\text{-E})$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \psi}}{\Gamma \vdash \phi \vee \psi}\ (\vee\text{-I}_2) \qquad \dfrac{\vdots}{\Gamma, \phi \vdash \eta} \qquad \dfrac{\vdots}{\Gamma, \psi \vdash \eta}}{\Gamma \vdash \eta}\ (\vee\text{-E})$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \phi} \qquad \dfrac{\vdots}{\Gamma \vdash \psi}}{\Gamma \vdash \phi \wedge \psi}\ (\wedge\text{-I})}{\Gamma \vdash \phi}\ (\wedge\text{-E}_1) \qquad \dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \phi} \qquad \dfrac{\vdots}{\Gamma \vdash \psi}}{\Gamma \vdash \phi \wedge \psi}\ (\wedge\text{-I})}{\Gamma \vdash \psi}\ (\wedge\text{-E}_2)$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \phi} \qquad x \notin \mathrm{FV}(\phi)}{\Gamma \vdash \forall x.\, \phi}\ (\forall\text{-I})}{\Gamma \vdash \phi[x \mapsto t]}\ (\forall\text{-E})$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \phi[x \mapsto t]}}{\Gamma \vdash \exists x.\, \phi}\ (\exists\text{-E}) \qquad \dfrac{\vdots}{\Gamma, \phi \vdash \psi} \qquad x \notin \mathrm{FV}(\Gamma, \psi)}{\Gamma \vdash \psi}\ (\exists\text{-E})$$

Figure 1.3: Listing of Cuts in Natural Deduction

Proof. By induction over the structure of the proof. Exercise : do the details. □

Corolary 1.7.2. *There is no cut-free proof of* $[] \vdash \bot$.

1.8 Cut elimination steps

It is possible to transform the proofs to get rid of the cuts. The first thing is to see that one can perform substitutions over natural deduction proofs.

Lemma 1.8.1 (weakening)**.** *Given a proof of* $\Gamma \vdash A$ *and a proposition* B *we have a proof of* $\Gamma; B \vdash A$.

Proof. The proof derivation is exactly the same, just keeping the extended context everywhere. □

Lemma 1.8.2. *Given a proof of* σ *of* $\Gamma; A \vdash B$ *and a proof* τ *of* $\Gamma \vdash A$ *we can construct a proof* $\sigma[A \backslash \tau]$ *of* $\Gamma \vdash B$ *which follows the structure of* σ *but replaces the uses of the axiom rule for* A *by copies of* τ.

Consider a proof ending with a cut:

$$\dfrac{\dfrac{\dfrac{\sigma}{\Gamma, A \vdash B}}{\Gamma \vdash A \implies B}\ (\implies\text{-I}) \quad \dfrac{\tau}{\Gamma \vdash A}}{\Gamma \vdash B}\ (\implies\text{-E})$$

We can erase this cut by rewriting the proof to:

$$\dfrac{\sigma[A \backslash \tau]}{\Gamma \vdash B}$$

Exercise

Describe the corresponding transformations erasing the other logical cuts (conjunction, disjunction, etc…)

Question

Why is it *not* obvious that the process of applying these transformation terminates ending with a cut-free proof ?

How can we show it terminates ?

Chapter 2

Higher-order logic

We present the formalism known as Higher-Order Logic (HOL). In this chapter I use HOL for the logical formalism, and not the proof-systems of the same name.

2.1 Naive Set Theory

There are two important novel features in HOL:

- The possibility to quantify over propositions and properties; one can also use the slogan "properties are objects of the formalism."
- The use of λ-calculus.

Indeed, HOL was designed by Alonzo Church shortly after he had invented λ-calculus. Furthermore, typed λ-calculus was invented *for* HOL. The introduction of types being a way to "repair" a first version of the formalism which was inconsistent. Here is a simple presentation of this paradox.

Set theories answer the question of the relation between the objects of a formalism and the properties by stipulating that every object, that is every set a, can be viewed as a property "being an element of a". In naive set theory, every property P can be turned into the set $\{x|(P\,x)\}$ of objects verifying P.

Going one step further, we can totally identify the set and the property. This means that:

- A set x is also a property,
- as a property that maps a set y to the proposition $y \in x$,
- this can be thus written $(x\,y)$ using the notation of λ-calculus.

But one can then encode Russell's paradox: the sets of sets which are not elements of themselves is encoded as:

$$R \equiv \lambda x.\neg(x\,x)$$

The property $R \in R$ then becomes:

$$(R\ R) = \lambda x.\neg(x\ x)\ \lambda x.\neg(x\ x)$$

which reduces to $\neg(R\ R)$.

In other words Russell's paradoxical construction is the fixpoint of negation. As such it is β-convertible to its negation, which entails inconsistency.

2.2 The objects of HOL

The objects of HOL are basically simply-typed λ-terms. More precisely:

Definition 2.2.1 (simple types)**.** Simple types are defined inductively by:

- There are two atomic simple types ι and o.
- If U and V are simple types, than $U \to V$ is a simple type.

The definition simply typed terms is the usual one. We can use a version where variables are tagged by their type.

Definition 2.2.2. The raw terms are built defined by the following grammar:

$$t \ ::= \ x^T \mid \lambda x^T.t \mid (t\ t).$$

The judgement $\vdash t : T$, meaning that t is of type T is defined inductively by the following, usual, rules:

$$\frac{}{\vdash x^T : T} \qquad \frac{\vdash t : T}{\vdash \lambda x^U.t : U \to T}$$

$$\frac{\vdash t : U \to T \qquad \vdash u : U}{\vdash (t\ u) : T}$$

The terms of type ι correspond to natural numbers and the terms of type o to propositions. Therefore we distinguish some special constants.

For ι:

$$\begin{array}{rcl@{\qquad}rcl} 0 & : & \iota & + & : & \iota \to \iota \to \iota \\ S & : & \iota \to \iota & \times & : & \iota \to \iota \to \iota \end{array}$$

and for o:

$$\begin{array}{rcl} \Longrightarrow & : & o \to o \to o \\ \forall_T & : & (T \to o) \to o \end{array}$$

Note that:

- We do not need other connectors (conjunction, disjunction…) at this stage.
- There is one quantifier $\forall_T$ for every type T. If P is a proposition, that is a term of type o depending of a variable x^T, then the proposition $\forall x^T.P$ will formally be constructed as $(\forall_T\ \lambda x^T.P)$.

2.3 The rules of HOL

Provability is defined in a similar way than for first-order logic; statements are sequents of the form $\Gamma \vdash A$ where A is a term of type o and Γ is a sequence of such terms.

$$\frac{}{[] \text{ wf}} \qquad \frac{\Gamma \text{ wf} \quad \vdash A : o}{\Gamma, A \text{ wf}}$$

$$\frac{\Gamma \text{ wf} \quad A \in \Gamma}{\Gamma \vdash A} \ (\mathsf{Ax})$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash (\Longrightarrow \ A\, B)} \ (\Longrightarrow \text{-I}) \qquad \frac{\Gamma \vdash (\Longrightarrow \ A\, B) \quad \Gamma \vdash A}{\Gamma \vdash B} \ (\Longrightarrow \text{-E})$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash (\forall_T \lambda x^T.A)} \ (\forall\text{-I}) \text{ if } x^T \text{ not free in } \Gamma \qquad \frac{\Gamma \vdash (\forall_T P) \quad \vdash t : T}{\Gamma \vdash (P\, t)} \ (\forall\text{-E})$$

$$\frac{\Gamma \vdash A \quad \vdash B : o}{\Gamma \vdash B} \ (\textsc{Conv}) \text{ if } A =_\beta B$$

2.4 Defining the missing connectors

As such, the language of HOL seems very poor: it contains the items of arithmetic, but lacks most of the logical connectors, at least as atomic constructs.

The answer is that it is possible to define the other connectors from $\forall$ and $\Longrightarrow$. The key is the very powerful ability to quantify over all propositions ($\forall_o$) to form a new proposition.

2.4.1 False proposition

The false proposition is the "less true" of all propositions. In HOL we can define:

$$\bot \equiv \forall_o \lambda X^o.X^o$$

Exercise. Check that for any $P : o$ one can prove $\bot \Longrightarrow P$.

2.4.2 Conjuction

$$A \wedge B \equiv \forall_o \lambda X^o.(A \Longrightarrow B \Longrightarrow X^o) \Longrightarrow X^o$$

2.4.3 Disjunction

$$A \vee B \equiv \forall_o \lambda X^o.(A \Longrightarrow X^o) \Longrightarrow (A \Longrightarrow X^o) \Longrightarrow X^o$$

2.4.4 Existential quantifier

$$\exists_T \equiv .\lambda P^{T\to o}.\forall X^o.(\forall_T x^T.(P\,x) \implies X^o) \implies x^T$$

2.5 Equality

The same idea allows to define properties and predicates; in general, we will then quantify over predicates and not just propositions. Foremost equality. Using quantification, one can state the fact that two objects, of the same type, verify the same properties. Like in Peano arithmetic, this is actually a characterization of being equal.

$$\begin{array}{lcl} =_T & : & T \to T \to o \\ =_T & \equiv & \lambda x^T.\lambda y^T.(\forall_{T\to o}\, \lambda P^{T\to o}.(P\,x^T) \implies (P\,y^T) \end{array}$$

In other words, we take advantage of the fact that we can now state Leibniz scheme to use it as a definition.

The point is that this is sufficient. In particular, we can prove that this relation is reflexive:

Exercise. Verifiy that for any type T, $\vdash \forall x^T.(=_T\ x^T\ x^T)$ is derivable.

One can also check that equality is indeed an equivalence relation:

Exercise. Verifiy that for any type T, the two following statements are derivable:

$$\vdash \forall x^T.\forall y^T.(=_T\ x^T\ y^T) \implies (=_T\ y^T\ x^T)$$

$$\vdash \forall x^T.\forall y^T.\forall z^T.(=_T\ x^T\ y^T) \implies .(=_T\ y^T\ z^T) \implies .(=_T\ x^T\ z^T)$$

2.6 Impredicativity

These two last exercises are a little trickier: they are solved by instantiating the predicate variable by a property involving equality itself. They therefore highlight the feature of HOL known as *impredicativity*: we can define a new proposition (resp. property) by quantifying over all propositions (resp. properties); that is the quantification includes the object which is being defined by the quantification.

This is a very powerful logical feature, which greatly enhances the power, or expressiveness of the formalism. For instance, it is possible to prove in higher-order arithmetic the consistency or Peano's arithmetic. What we see here is that it also is a very flexible, compact and convenient way to define logical constructions.

2.7 Examples of impredicative encodings

An important general scheme is that the impredicative quantification allows to define a predicate as the smallest property closed by a finite set of clauses. We here give some examples, writing the clauses in Coq Syntax.

Equality

We have already given the impredicative definition above. It actually corresponds, given a type T and an object $x : T$ to define the property "being equal to x" as the smallest property verified by x:

```
Inductive equal (T : Type)(x : T) : T -> Prop :=
  refl_equal : equal T x x.
```

Even numbers

The set of even numbers can be defined as the smallest set containing 0 and closed by the operation $+2$. In Coq syntax:

```
Inductive even : nat -> Prop :=
| even0 : even 0
| evenS : forall n, even n -> even (S (S n)).
```

The impredicative encoding is, as for equality, the elimination/induction scheme over the numbers verifying the property:

$$\text{even} \equiv \lambda x^{\iota}.\forall P^{\iota\to o}.(P\,0) \implies (\forall n^{\iota}.(P\,n) \implies (P\,(S\,(S\,n^{\iota}))) \implies (P\,x^{\iota}).$$

Greater or equal

The usual relation $n \leq m$ can be defined inductively. More precisely, given n, it is the set of numbers greater than n which is defined as the set containing n and closed by successor:

```
Inductive le (n : nat) : nat -> Prop :=
 | le_n : le n n
 | le_S : forall m : nat, le n m -> le n (S m).
```

In HOL :

$$(\text{le}\,n) \equiv \lambda m^{\iota}.\forall P^{\iota\to o}.(P\,n) \implies (\forall x^{\iota}.(P\,x^{\iota}) \implies (P\,(S\,(S\,x))) \implies (P\,m^{\iota}).$$

well-founded relations

A less intuitive but useful definition is well-foundedness. Given a relation R, we say that x is accessible if there is no infinite path starting from x; that is no $x_1, x_2, ...$ such that

$$(R\,x\,x_1), (R\,x_1\,x_2), ...$$

We can define the accessibility predicate inductively as:

```
Inductive Acc (T:Type)(R:T->T->Prop) :=
  Acc_i : forall x, (forall y, R x y -> Acc y) -> Acc x.
```

2.8 Arithmetic in HOL

In order to have all of arithmetic in HOl, we need some axioms.

Two axioms of arithmetic have to be assumed: $\forall x, (S\, x) \neq 0$ and the injectivity of the successor.

Then In HOL, we can state the induction scheme as a proposition. We have two ways to handle it:

- We can add it as an axiom,
- or we can decide that when we translate a proposition of arithmetic to HOL, we always bound quantification to numbers verifying the induction scheme.

The second solution means that we define the predicate, for natural numbers, corresponding to the induction scheme:

$$\mathsf{Nat} \equiv \lambda n^{\iota}.\forall P^{\iota\to o}.(P\, 0) \implies (\forall x^{\iota}.(P\, x) \implies (P\, (S\, x^{\iota})) \implies (P\, n^{\iota}).$$

So $(\mathsf{Nat}\, n)$ means we can apply the induction principle to n.

The proposition of arithmetic $\forall x, x + 0 = x$ is then translated to the following, provable, HOL proposition:

$$\forall x^{\iota}.(\mathsf{Nat}\, n^{\iota}) \implies n^{\iota} + 0 = n^{\iota}$$

2.9 Functions in HOL

Simply typed λ-calculus is not powerful as a programming language. Therefore, in HOL, to define new functions, we have to prove their existence. Consider the predecessor function; we can prove:

$$\forall x^{\iota}, \exists y^{\iota}, x^{\iota} = 0 \vee x = (S\, y^{\iota})$$

The proof is easy, by induction over x^{ι}.

In order to be able to name the function which maps x to y, it is common practice to extend the language of HOL by a description operator, also called Hilbert's ε operator.

2.9.1 Principle of Hilbert's epsilon in FOL

Originally, Hilbert's operator was an alternative to the existential quantifier. The idea is that:

- For every property P we have an object $\varepsilon(P)$,
- this object is "the first" object to verify the property P if such an object exists. Thus, for any x, $P(x) \implies P(\varepsilon(P))$.

In other words, logically it is equivalent to have $\exists x.P(x)$ and $P(\epsilon(P))$. But ε gives us a way to *name* the object verifying P.

2.9.2 Hilbert's epsilon in HOL

In HOL, we have to take typing into account. We have one operator for every type and property over this type. Actually, we can type the operator as mapping an object to every property. For every type T:

$$\vdash \varepsilon_T : (T \to o) \to T$$

We then just need to add one primitive logical rule corresponding to the

$$\frac{\Gamma \vdash (P\, t) \quad \vdash P : T \to o}{\Gamma \vdash (P\, (\varepsilon_T\, P))} \ (\varepsilon)$$

2.9.3 Using epsilon to define functions

A simple example: we want to define the function div2 which maps x to the integer quotient of x divided by 2. We first prove:

$$\forall_\iota x, \exists_\iota y, x = y + y \vee x = S(y + y)$$

which is done by induction over x.

We then can define:

$$(div2) = \lambda x.(\varepsilon_\iota\ \lambda y.x = y + y \vee x = S(y + y))$$

and prove, using the rule (ε) that:

$$\forall x, x = (\mathsf{div2}\ x) + (\mathsf{div2}\ x) \vee x = (S\ (\mathsf{div2}\ x) + (\mathsf{div2}\ x)).$$

This mechanism allows to define many functions in HOL. It also can be added to first-order logic in order to define functions in first-order arithmetic. It is however important to notice that these functions do *not* come with new reductions. We do not, for instance, have the following reductions:

$$\begin{array}{rcl} (\mathsf{div2}\ 0) & \rhd^* & 0 \\ \mathsf{div2}\ (S\ (S\ 0)) & \rhd^* & (S\ 0) \end{array}$$

This is different from what happens in (Coq's) type theory where we can define a function bearing these reductions.

2.9.4 Epsilon and classical logic

This is a somewhat more advanced topic, but interesting.

Adding the epsilon operator to intuitionistic logic makes the logic almost classical.

Here is a sequence of exercises illustrating this point.

Exercise

Check that in intuitionistic arithmetic, one can prove:

$$\forall x, \forall y, x = y \vee x \neq y.$$

Exercise

We say that a proposition P is decidable when $P \vee \neg P$ is provable. Show that when A and B are decidable, then so are $A \vee B$, $A \wedge B$ and $A \implies B$.

Exercise

We now may use the epsilon operator. Show that is for any object t, the proposition $P[x \backslash t]$ is decidable, then $\exists x.P$ is decidable.

Exercise

Under the same assumptions, show that $\forall x, P$ is decidable.

Exercise

What can you deduce about Heyting arithmetic equipped with the epsilon operator ?

Chapter 3

Normalization proofs

3.1 Principles

In all type systems, the judgement $\vdash t : T$ or $\Gamma \vdash t : T$ is defined inductively, and the typing derivation closely follows the structure of the term t. Normalization will be proved by induction over t or over the typing derivation, both formulations being equivalent.

The main difficulty is thus to formulate the correct induction hypothesis. One can check that it strong normalization alone is not sufficient, because of the application case: if t and u are SN, $(t\,u)$ is not necessarily SN. We thus need to have an induction hypothesis which (1) depends of the type and (2) is stronger for function types in order to treat the application case.

3.2 Simply typed lambda-calculus

Following the idea above, we come up with a simple solution. For any type T we define a set $|T|$ of terms *reducible for* T. This set is defined recursively over T:

- $|A| \equiv$ SN if A is atomic (in particular, for HOL, $|\iota| = |o| =$ SN).
- $|U \to T| \equiv \{t \in \Lambda, \forall u \in |U|, (t\,u) \in |T|\}$

The main plan is thus:

- To show, by induction over t, that if $\vdash t : T$ then $t \in |T|$.
- On the other hand, that if $t \in |T|$ then $t \in$ SN, which means that we have indeed strengthened the induction hypothesis.

Some well-chosen properties will be useful for both parts.

Definition 3.2.1 (Atomic terms)**.** a term is said to be atomic if it is strongly normalizable and of the form $(x\,t_1\,t_2\,\ldots\,t_n)$. We write $\mathcal{A}$ for the set of atomic terms.

Lemma 3.2.1. *For every type T the three following properties hold:*

1. $|T| \subset$ SN.
2. *All atomic terms belong to* $|T|$*:* $\mathcal{A} \subset |T|$.
3. *If* $(t[x\backslash u]\ u_1\ \ldots\ u_n) \in |T|$ *and* u *is strongly normalizing, then* $(\lambda x.t\ u\ u_1\ \ldots\ u_n) \in |T|$.

The last clause states that $|T|$ is also closed by a form of β-expansion. Note also that the second clause entails that $|T|$ is not empty.

Proof. The three assertions are proved together by induction over T.

If T is atomic then $|T| =$ SN. So (1) is true by construction. It is also straightforward that SN verifies (2) and quite easy to check that it verifies (3).

If T is of the form $U \to V$, we know by induction hypothesis that $|U|$ and $|V|$ verify (1), (2) and (3). We can the prove:

(1) Take $t \in |U \to V|$. Since $|U|$ verifies (3), we know that, for instance, any variable x is in $|U|$. So $(t\ x) \in |V|$. Because of (1), $|V| \subset$ SN, so $(t\ x) \in$ SN and thus $t \in$ SN.

(2) Take $t = (x\ t_1\ t_2\ \ldots\ t_n) \in \mathcal{A}$ and $u \in |U|$. We want to prove $t \in |U \to V|$, which means checking that $(t\ u) \in |V|$. But we see that $(t\ u) = (x\ t_1\ t_2\ \ldots\ t_n\ u) \in \mathcal{A} \subset |V|$ because $|V|$ verifies (2).

(3) Take $w = (t[x\backslash u_0]\ u_1\ \ldots\ u_n) \in |U \to V|$ with $u_0 \in$ SN. We need to prove that $(\lambda x.t\ u_0\ u_1\ \ldots u_n) \in |U \to V|$.

For any $u \in |U|$, we know that $(w\ u) = (t[x\backslash u_0]\ u_1\ \ldots\ u_n\ w)\|V|$. Thus, because $|V|$ verifies (3) we have indeed $(\lambda x.t\ u_0\ u_1\ \ldots u_n\ u) \in |V|$. □

Definition 3.2.2 (correct valuation)**.** We call valuation a function $\mathcal{J}$ which associates a λ-term to each variable. We say that this valuation is correct when for every variable x^T we have:

$$\mathcal{J}(x^T) \in |T|.$$

We write $|t|_{\mathcal{J}}$ for the term t where very free variable has been substituted by its image through $\mathcal{J}$.

We can state and prove the main lemma:

Lemma 3.2.2. *If* $\vdash t : T$*, then for any correct valuation* $\mathcal{J}$*, we have:* $|t|_{\mathcal{J}} \in |T|$.

Proof. We reason by induction over t.

- If t is of the form x^T, then $|x^T|_{\mathcal{J}} = \mathcal{J}(x^T)$ which belongs to $|T|$ because $\mathcal{J}$ is correct.
- If t is of the form $(u\ v)$, we know that $u : V \to T$ and $v : V$. So $|u|_{\mathcal{J}} \in |V \to T|$ and $|v|_{\mathcal{J}} \in |V|$. Thus $|(u\ v)|_{\mathcal{J}} = (|u|_{\mathcal{J}}\ |v|_{\mathcal{J}}) \in |T|$.
- The most tricky case is when t is of the form $\lambda x^V.u$; then we know that $T = V \to U$ and $u : U$. Also, by induction, $|u|_{\mathcal{J}} \in |U|$.

  Given any $v \in |V|$, we need to show that $(|\lambda x^V.u|_{\mathcal{J}}\ v) \in |U|$. We do so by induction over the number of possible successive reductions starting from v.

Because $(|\lambda x^V.u|_\mathcal{J}\ v)$ is neutral, we just need to show that all its reducts are in
We have: $|\lambda x^V.u|_\mathcal{J} = \lambda x^V.|u|_{\mathcal{J};x^V\leftarrow x^V}$ and thus

$$(|\lambda x^V.u|_\mathcal{J}\ v)\ ...$$

□

3.3 Proof variants

There are several variants of the proof above. In particular, there are alternative formulations to the lemma 3.2.1 which are similar but not exactly equivalent. A quite elegant one is the one of Girard [1].

Definition 3.3.1 (Neutral terms)**.** A term is said to be neutral if it is not of the form $\lambda x^T.t$ (or equivalently if it is an application or a variable). One writes $\mathcal{N}$ for the set of neutral terms.

One can note that when a term u is neutral, then there are no new redexes created by a substitution $t[x\backslash u]$.

Lemma 3.3.1. *For any type T, the set $|T|$ verifies the following conditions:*

1. $|T| \subset$ SN.
2. *$|T|$ is closed by reduction:* $\forall t \in |T|, t \rhd_\beta t' \implies t' \in |T|$.
3. *If $t \in \mathcal{N}$ and*
$$\forall t', t \rhd_\beta t' \implies t' \in |T|$$
then $t \in |T|$.

Again, the third conditions is a closure property by a (slightly different) form of β-expansion. One can also notice that a consequence of (3) is:

Remark. Every atomic strongly normalizing term is in $|T|$.

Again, the three closure conditions have to be proved together/

Proof. By induction over (the structure of) T.

If T is atomic, (1) is trivial by definition. Conditions (2) is also obvious: SN is closed by reduction. Finally of all reducts of t are SN, then t is SN, so (3) holds (we do not use the condition $t \in \mathcal{N}$ here).

If $T = U \to V$ with $|U|$ and |V| verifying the three closure conditions:

(1) Because $|U|$ verifies (3), we know that any $x \in |U|$, so for every $t \in |U \to V|$ we have $(t\ x) \in |V|$. Because $|V|$ verifies (1) we know that $(t\ u)$ is SNand thus $t \in$ SN.

(2) If $t \in |U \to V|$ and $t \rhd_\beta t'$, then for any $u \in |U|$ we know that $(t\ u) \in |V|$. Because $|V|$ verifies (2) we also have $(t'\ u) \in |V|$. Thus $t' \in |U \to V|$.

(3) Suppose that $t \in \mathcal{N}$ and any reduct of t' of t is element of $|U \to V|$. Consider $u \in |U|$; we need to prove $(t\ u) \in |V|$.

Because $(t\ u)$ is neutral, and $|V|$ verifies (3) it suffices to check that any reduct of $(t\ u)$ is in $|V|$. We reason by induction over the number of possible consecutive reduction steps stating from u (we know that u is SN because $|U|$ verifies (2)). Because t is neutral, the term $(t\ u)$ can only reduce to:

- $(t'\ u)$ with $t \rhd_\beta t'$, but then $t' \in |U \to V|$ and thus $(t'\ u) \in |V|$.
- $(t\ u')$ with $u \rhd_\beta u'$, but then the number of consecutive reduction steps in the argument has decreased.

□

This lemma allows to show that typing entails reducibility. In other words, we can give another proof of lemma 3.2.2, with little technical differences.

Definition 3.3.2. When t is a strongly normalizing term, we call $\mu(t)$ the length of the longest reduction path starting from t.

In particular when t is normal, then $\nu(t) = 0$ and when $t \rhd_\beta t'$ then $\nu(t') < \nu(t)$.

Lemma 3.3.2. *Given types U and V and term t, if for any term $u \in |U|$ we have $t[x^U \backslash u] \in |V|$, then*

$$\lambda x^U.t \in |U \to V|.$$

Proof. Consider $u \in |U|$, we prove by induction over $\mu(u) + \mu(t)$ that $(\lambda x^U.t\ u) \in |V|$. The term can reduce to:

- $t[x^U \backslash u]$ which is in $|V|$ by hypothesis.
- $(\lambda x^U.t\ u')$ with $u \rhd_\beta u'$. In this case we know that $u' \in |U|$ and $\mu(u') < \mu(u)$ so we can use the induction hypothesis.
- $(\lambda x^U.t'\ u)$ with $t \rhd_\beta t'$. In this case we know that for all $u' \in |U|$ $t'[x^U \backslash u'] \in |V|$ and $\mu(t') < \mu(t)$ so we can use the induction hypothesis.

□

Proof. The cases where t is a variable or an application are easy and identical to the original proof. The interesting case is when t is of the form $\lambda x^V.u : V \to U$.

We have $|\lambda x^V.u|_{\mathcal{J}} = \lambda x^V.|u|_{\mathcal{J};x^V \leftarrow x^V}$ so we need to show that: $\lambda x^V.|u|_{\mathcal{J};x^V \leftarrow x^V} \in |V \to U|$.

Using previous lemma, this boils down to proving, for any $v \in |V|$:

$$(|u|_{\mathcal{J};x^V \leftarrow x^V})[x^V \backslash v] \in |U|.$$

The properties of substitution entail $(|u|_{\mathcal{J};x^V \leftarrow x^V})[x^V \backslash v] = |u|_{\mathcal{J};x^V \leftarrow v}$.

Because $\mathcal{J}; x^V \leftarrow v$ is a correct interpretation, this is indeed a consequence of the induction hypothesis for u.

□

3.4 Normal terms in HOL

A normal λ-term is always of the form: $\lambda x_1.\ ...\ \lambda x_n.(x\ u_1\ ...\ u_m)$.

In HOL, we have special variables corresponding to the constructs of arithmetic:

$$\begin{aligned} O &: \iota \\ S &: \iota \to \iota \\ + &: \iota \to \iota \to \iota \\ \times &: \iota \to \iota \to \iota \\ =_T &: (T \to o) \to o \end{aligned}$$

A closed normal term of type ι can be:

- 0
- $(S\ t)$ where t is itself normal, closed of type ι,
- $(+\ t\ u)$ or $(\times\ t\ u)$ with t and u themselves normal, closed of type ι.

We see these terms correspond to a constant.

If we add a variable $x : \iota$, the terms which can be constructed are x, constants, and is closed by addition and multiplication. In other words, they correspond to a polynoms : $\Sigma_{i=0}^{k} a_i \cdot x^i$.

A closed normal term of type $\iota \to \iota$ is either S, $(+\ t)$, $(\times\ t)$ or of the form $\lambda x.t$ with $t : \iota$. In other words:

Lemma 3.4.1. *If $t : \iota \to \iota$ in simply typed λ-calculus and is closed, then it corresponds to a polynomial.*

3.5 System T

Gödel's System was introduced in 1958 in order to study cut elimination in arithmetic. We here study it on one hand because it allows to prove cut elimination for the kernel of Martin-Löf's type theory, but also because it is the kernel of the functional terminating fragment of ML which are the basic objects of Coq.

System T is simply typed λ-calculus enriched by an operator allowing simple case analysis and strutural recursion of unary natural numbers.

The additional objects are:

- A constant $0 : \iota$,
- a constant $S : \iota \to \iota$,
- for every type T an operator $R_T : \iota \to T \to (\iota \to T \to T) \to T$.

Furthermore, β-reduction is enriched by two rewriting rules:

$$\begin{array}{rcl}(R\,0\,t_0\,t_S) & \rhd & t_0 \\ (R\,(S\,t)\,t_0\,t_S) & \rhd & (t_S\,t(R\,t\,t_0\,t_S)\,)\end{array}$$

The normalization proof is similar to simply typed λ-calculus, but we have to take the new reductions into account.

The normalization proof is similar to simply typed calculus.

Definition 3.5.1. For every typed T, the set of reducible terms is defined by:

$$\begin{array}{rcl}|\iota| & = & \mathsf{SN} \\ |A \to B| & = & \{t, \forall u \in |A|, (t\,u) \in |B|\}\end{array}$$

Lemma 3.5.1. *For every type T, the following propositions hold:*

1. $|T| \subset$ SN
2. $\mathcal{A} \subset |T|$
3. (a) *If* $(t[x\backslash u]\ u_1\ \ldots\ u_n) \in |T|$ *and* u *is strongly normalizing, then* $(\lambda x.t\,u\,u_1\ \ldots\ u_n) \in |T|$.
   (b) *If* $t_0 \in |T|$ *and* $t_S \in$ SN, *then* $(R\,0\,t_0\,t_S) \in |T|$.
   (c) *If* $t_S\,t\,(R\,t\,t_0\,t_S) \in |T|$, *then* $(R\,(S\,t)\,t_0\,t_S) \in |T|$.

The rest of the proof follows precisely the one of simply-typed λ-calculus.

3.6 System F

$$\begin{array}{rcl}T & ::= & \alpha \mid T \to T \mid \forall\alpha.T \\ t & ::= & x \mid \lambda x.t \mid (t\,t)\end{array}$$

$$\ll Var\frac{}{\Gamma \vdash x : T}\ (\text{If } (\mathrm{x} : \mathrm{T}) \in \Gamma)$$

$$\ll App\frac{\Gamma \vdash t : U \to T \quad \Gamma \vdash u : U}{\Gamma \vdash (t\,u) : T} \qquad \ll Lam\frac{\Gamma(x : U) \vdash t : T}{\Gamma \vdash \lambda x.t : U \to T}$$

$$\ll Fa\frac{\Gamma \vdash t : T}{\Gamma \vdash t : \forall\alpha.T}\ (\text{If } \alpha \text{ not free in } \Gamma) \qquad \ll Inst\frac{\Gamma \vdash t : \forall\alpha.T}{\Gamma \vdash t : T[\alpha\backslash U]}$$

Definition 3.6.1 (Reducibility candidate)**.** A set $\mathcal{CR}$ of λ-terms is a reducibility candidate if and only if it verifies the three closure properties:

1. $\mathcal{C} \subset$ SN
2. $\forall t \in \mathcal{A} \cap$ SN$, t \in \mathcal{C}$

3. $(t[x\backslash u]\, u_1\ \ldots\ u_n) \in \mathcal{C} \wedge u \in \mathsf{SN} \implies (\lambda x.t\, u\, u_1\ \ldots\ u_n) \in \mathcal{C}$

We call $\mathcal{CR}$ the set of reducibility candidates.

Lemma 3.6.1. *The set* SN *of strongly normalizing terms is a reducibility candidate. So $\mathcal{CR}$ is not empty.*

Lemma 3.6.2. *Given any family $(\mathcal{C}_i)_{i\in I}$ of reducibility candidates, if I is not empty, then the intersection is a reducibility candidate: $\bigcap_{i\in I} \mathcal{C}_i \in \mathcal{CR}$.*

Definition 3.6.2. For any type variable α, $I(\alpha)$ is a reducibility candidate. Then, for any type T we define a set of λ-terms $|T|_{\mathcal{J}}$ by:

$$\begin{aligned} |\alpha|_{\mathcal{J}} &= I(\alpha) \\ |U \to V|_{\mathcal{J}} &= \{t, \forall u \in |U|_{\mathcal{J}}, (t\, u) \in |V|_{\mathcal{J}}\} \\ |\forall \alpha.T|_{\mathcal{J}} &= \bigcap_{\mathcal{C}\in\mathcal{CR}} |T|_{\mathcal{J};\alpha\leftarrow\mathcal{C}} \end{aligned}$$

An easy technical lemma is that the interpretation of types commutes with substitution:

Lemma 3.6.3. *For all types T and U and type variable α we have*

$$|T[\alpha\backslash U]|_{\mathcal{J}} = |T|_{\mathcal{J};\alpha\leftarrow|U|_{\mathcal{J}}}.$$

The important lemma, who's proof is very similar to the previous cases:

Lemma 3.6.4. *For every type T, $|T|_{\mathcal{J}}$ is a reducibility candidate.*

Lemma 3.6.5. *Suppose that for every term variable x, if $(x : U) \in \Gamma$ then $\mathcal{J}(x) \in |U|_{\mathcal{J}}$. Then, when $\Gamma \vdash t : T$ holds, we have $|t|_{\mathcal{J}} \in |T|_{\mathcal{J}}$.*

Proof. In this version of system F, where abstractions do not carry types, there is no perfect isomorphism between the term and its typing derivation[1]. So formally, it is important to reason by induction over the typing derivation.

The proof however is very similar to the ones for the systems above.

Var We know by hypothesis that $\mathcal{J}(x) \in |T|_{\mathcal{J}}$.

App By induction, $|t|_{\mathcal{J}} \in |U \to T|_{\mathcal{J}}$ and $|u| \in |U|_{\mathcal{J}}$, so $|(t\, u)|_{\mathcal{J}} \in |T|_{\mathcal{J}}$.

Lam For any $u \in |U|_{\mathcal{J}}$, we know that $|t|_{\mathcal{J};x\leftarrow u} \in |T|_{\mathcal{J}}$, which means that $|t|_{\mathcal{J};x\leftarrow x}[x\backslash u] \in |T|_{\mathcal{J}}$. Because $|T|_{\mathcal{J}}$ is a reducibility candidate, this entails that $(\lambda x.|t|_{\mathcal{J};x\leftarrow x}\ u) \in |T|_{\mathcal{J}}$ and thus $\lambda x.|t|_{\mathcal{J};x\leftarrow x} \in |U \to T|_{\mathcal{J}}$. This last statement implying $|\lambda x.t|_{\mathcal{J}} \in |U \to T|_{\mathcal{J}}$.

Fa Take $\mathcal{J}$ and $\mathcal{J}$ such that if x is bound to A in $|\Gamma|$ then $\mathcal{J}(x) \in |A|_{\mathcal{J}}$. Since α is not free in A, we know that for any $\mathcal{C} \in \mathcal{CR}$ we also have $I(x) \in |A|_{\mathcal{J};\alpha\leftarrow\mathcal{C}}$. Thus $|t|_{\mathcal{J}} \in |T|_{\mathcal{J};\alpha\in\mathcal{C}}$ and also $|t|_{\mathcal{J}} \in \cap_{\mathcal{C}\in\mathcal{CR}}|T|_{\mathcal{J};\alpha\in\mathcal{C}}$.

[1] This is called the "Curry style presentation" of system F, as opposed to the "Church style".

Inst We know that $|t|_{\mathcal{I}} \in \cap_{\mathcal{C} \in \mathcal{CR}} |T|_{\mathcal{J};\alpha \in \mathcal{C}}$ and thus, in particular, that $|t|_{\mathcal{I}} \in |T|_{\mathcal{J};\alpha \in |U|_{\mathcal{J}}}$. The latter is equivalent to $|t|_{\mathcal{I}} \in |T[\alpha \backslash U]_{\mathcal{J}}$.

□

Corolary 3.6.6. *If* $\Gamma \vdash t : T$*, then* $t \in$ SN.

Chapter 4

Dependent Types

4.1 Definition

$$\begin{array}{rcl} s & ::= & \mathsf{Kind} \mid \mathsf{Prop} \\ t & ::= & x \mid \lambda x : t.t \mid (t\ t) \mid \forall x : t.t \mid s \end{array}$$

$$\frac{}{[]\ \mathsf{wf}} \qquad \frac{\Gamma \vdash A : s}{\Gamma(x : A)\ \mathsf{wf}}$$

$$\frac{\Gamma\ \mathsf{wf}}{\Gamma \vdash \mathsf{Prop} : \mathsf{Kind}} \qquad \frac{\Gamma\ \mathsf{wf}}{\Gamma \vdash x : A}\ (\mathsf{if}\ (\mathrm{x} : \mathrm{A}) \in \Gamma)$$

$$\frac{\Gamma \vdash t : \forall x : A.B \quad \Gamma \vdash u : A}{\Gamma \vdash (t\ u) : B[x \backslash u]} \qquad \frac{\Gamma(a : A) \vdash t : B \quad \Gamma \vdash \forall x : A.B : s}{\Gamma \vdash \lambda x : A.t : \forall x : A.B}$$

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma(x : A) \vdash B : s_1}{\Gamma \vdash \forall x : A.B : s_1}\ (\mathsf{if}\ (\mathrm{s_1}, \mathrm{s_2}) \in \mathcal{R}) \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash B : s}{\Gamma \vdash t : B}\ (\mathsf{if}\ \mathrm{A} =_\beta \mathrm{B})$$

The type system is parametrized by $\mathcal{R}$.

If $\mathcal{R} = \{(\mathsf{Prop}, \mathsf{Prop})\}$ then it is the simply typed calculus.

If we add $(\mathsf{Kind}, \mathsf{Prop})$ we get impredicativity. $\mathcal{R} = \{(\mathsf{Prop}, \mathsf{Prop}); (\mathsf{Kind}, \mathsf{Prop})\}$ gives the system F.

If we add $(\mathsf{Prop}, \mathsf{Kind})$ we have dependent types. $\mathcal{R} = \{(\mathsf{Prop}, \mathsf{Prop}); (\mathsf{Prop}, \mathsf{Kind})\}$ gives the LF (logical framework).

If we have all the rules, $\mathcal{R} = \{(\mathsf{Prop}, \mathsf{Prop}); (\mathsf{Prop}, \mathsf{Kind}); (\mathsf{Kind}, \mathsf{Prop}); (\mathsf{Kind}, \mathsf{Kind})\}$ we have defiend the calculus of constructions (CoC) which is the core of Coq.

4.2 Basic properties

Lemma 4.2.1 (Substitution)**.** *If* $\Gamma(x : A)\Delta \vdash t : T$ *and* $\Gamma \vdash u : A$ *then* $\Gamma(\Delta[x \backslash u]) \vdash t[x \backslash u] : T[x \backslash u]$.

Lemma 4.2.2 (weakening)**.** *If* $\Gamma \vdash t : T$ *and* Γ' wf *are derivable, and* $\Gamma \subset \Gamma'$ *(meaning that* Γ *is a subsequence of* Γ'*) then* $\Gamma' \vdash t : T$*.*

Lemma 4.2.3. *If* $\Gamma \vdash t : T$ *is derivable, then* Γ wf *is derivable.*

Lemma 4.2.4. *If* $\Gamma \vdash t : T$ *is derivable, then either* $T =$ Kind *or* $\Gamma \vdash T : s$ *is derivable for some sort* s*.*

Lemma 4.2.5. *If* $\Gamma :\vdash \forall x : A.B : s$ *then* $\Gamma(x : A) \vdash B : s$*.*

Lemma 4.2.6 (Inversion)**.** *If* $\Gamma \vdash x : A$ *then there exists* A' *such that* $(x : A') \in \Gamma$ *and* $A' =_\beta A$*.*

If $\Gamma \vdash (t\ u) : A$ *then there exist* B *and* C *such that* $A =_\beta C[x\backslash u]$*,* $\Gamma \vdash u : B$ *and* $\Gamma \vdash t : \forall x : B.C$*.*

If $\Gamma \vdash \lambda x : A.t : B$ *then there exists* C *such that* $B =_\beta \forall x : A.C$*,* $\Gamma(x : A) \vdash t : C$ *and* $\Gamma \vdash \forall x : A.C : s$*.*

If $\Gamma \vdash forall x : A.B : T$ *then* T *is either* Prop *or* Kind *and* $\Gamma(x : A) \vdash B : T$*.*

If $\Gamma \vdash$ Prop $: T$ *then* $T =$ Kind*.*

Corolary 4.2.7 (Type uniqueness)**.** *If* $\Gamma \vdash t : A$ *and* $\Gamma \vdash t : B$ *are derivable, then* $A =_\beta B$*.*

Lemma 4.2.8 (Subject reduction)**.** *If* $\Gamma \vdash t : A$ *and* $t \rhd_\beta t'$ *(resp.* $A \rhd_\beta A'$*) then* $\Gamma \vdash t' : A$ *(resp.* $\Gamma \vdash t : A'$*).*

4.3 Type checking

A key result is, of course, strong normalisation.

Theorem 4.3.1. *If* $\Gamma \vdash t : T$*, then* t *and* T *are strongly normalizing.*

We sketch the proof of the theorem later. An important point is that normalization entails decidability of β-conversion; and this, using the inversion lemma above, entails decidability of type-checking.

Theorem 4.3.2. *Given* Γ *and* t*, the following propositions are decidable:*

- Γ wf
- *There exists* T *such that* $\Gamma \vdash t : T$*.*

Proof. The proof, like the algorithm, proceeds by induction over the structure of t. In every case, one uses the corresponding clause of the inversion lemma. We only detail key cases:

- If $t = x$ then one checks that Γ is well-formed and that x is bound in Γ. If so, x has the corresponding type; if not, there is no type.
- If $t = (u\ v)$. Then one checks that u and v have types: $\Gamma \vdash u : U$ and $\Gamma \vdash V$. On then checks that U reduces to some function type: $U \rhd^*_\beta \forall x : V'.W$. If it is the case, one checks that $V =_\beta V'$, in which case $\Gamma \vdash t : W[x\backslash v]$. In all other cases, t is not well-typed.

- To check that $\Gamma(x : A)$ is well-formed, one checks whether there exists T such that $\Gamma \vdash A : T$, and then that T is a sort.

□

4.4 Erasing the dependency

Given a λ-term with dependent types, we can type the same λ-term erasing the dependent types; that is with a simpler type.

In order to define this transformation, we write:

- K for kinds, that is terms of type Type. Note that Prop is a kind.
- T for predicates, that is terms of a kind. Note this includes types, that is terms of type Prop. We also write greek letters (α, β for variables that are predicates).
- $t, u \ldots$ for proofs, that is terms whose types is of type Prop (or equivalently whose type is a predicate / type). We also write $x, y \ldots$ for variables that are proofs.

We define a transformation over predicates and kinds: if T, resp. K is a predicate, resp. a kind, then so is $\overline{T}$, resp. $\overline{K}$.

For kinds:

$$\begin{array}{rcl} \overline{\mathsf{Type}} & = & \mathsf{Type} \\ \overline{\forall x : T.K} & = & \overline{K} \\ \overline{\forall \alpha : K_1.K_2} & = & \forall \alpha : \overline{K_1}.\overline{K_2} \end{array}$$

For predicates:

$$\begin{array}{rcl} \overline{\forall x : T_1.T_2} & = & \overline{T_1} \to \overline{T_2} \\ \overline{\forall \alpha : K.T} & = & \forall \alpha : \overline{K}.\overline{T} \\ \overline{\alpha} & = & \alpha \\ \overline{(T\,t)} & = & \overline{T} \\ \overline{\lambda x : U.T} & = & \overline{T} \\ \overline{(T\,U)} & = & (\overline{T}\,\overline{U}) \\ \overline{\lambda \alpha : K.T} & = & \lambda : \overline{K}.\overline{T} \end{array}$$

We extend this for contexts:

$$\begin{array}{rcl} \overline{[]} & = & [] \\ \overline{\Gamma(x : T)} & = & \overline{\Gamma}(x : \overline{T}) \\ \overline{\Gamma(\alpha : K)} & = & \overline{\Gamma}(\alpha : \overline{K}) \end{array}$$

Lemma 4.4.1. *If* $\Gamma \vdash T : K$ *or resp.* $\Gamma \vdash K :$ Kind*, then* $\overline{\Gamma} \vdash \overline{T} : \overline{K}$*, resp.* $\overline{\Gamma} \vdash \overline{K} :$ Kind*.*

We do not detail the proof here.

Transformation for proofs:

$$\begin{array}{rcl} \overline{x} & = & x \\ \overline{\lambda x : T.t} & = & \lambda x : \overline{T}.\overline{t} \\ \overline{(t\,u)} & = & (\overline{t}\,\overline{u}) \\ \overline{\lambda \alpha : K.t} & = & \lambda \alpha : \overline{K}.\overline{t} \\ \overline{(t\,T)} & = & (\overline{t}\,\overline{T}) \end{array}$$

Lemma 4.4.2. *If* $\Gamma \vdash t : T$ *then* $\overline{\Gamma} \vdash \overline{t} : \overline{T}$*.*

Note that we do not treat primitive inductive types here. Actually, we can check this transformation does not work for the whole of Coq.

We recall that we can define natural numbers, false and equality using just λ-terms in the Calculus of Constructions:

$$\begin{array}{rcl} \mathsf{nat} & = & \forall \alpha : \mathsf{Type}.\alpha \to (\alpha \to \alpha) \to \alpha \\ (\mathsf{eq\ A\ a\ b}) & = & \forall \alpha : A \to \mathsf{Prop}.(\alpha\, a) \to (\alpha\, b) \end{array}$$

Exercise

What are the images of nat and eq through the transformation defined above ?

Exercise

Show that $0 \neq 1$ is not provable in the Calculus of Constructions.

Can you see what is the feature of Coq that allows to show $0 \neq 1$ and which is not present in CoC ?

Bibliography

[1] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[2] J.-L. Krivine. *Théorie des ensembles*. Nouvelle bibliothèque mathématique. Cassini, 1998.

[3] J. van Heijenoort. *From Frege to Gödel, A Source Book in Mathematical Logic, 1879–1931*. Source Books in the History of Science. Harvard University Press, 1967.