MPRI

Foundations of Proof Systems

Benjamin Werner
Dominik Kirst

2026-2027

Contents

Chapter 1

Inductive definitions

Many definitions in logic are inductive definitions. This means defining a set (resp. type) as being the smallest set (resp. type) verifying some closure conditions.

Examples are:

The set of natural numbers $\mathbb{N}$ *is the smallest set such that:*

- $0 \in \mathbb{N}$,
- *if* $n \in \mathbb{N}$, *then* $S(n) \in \mathbb{N}$.

The set of even *natural numbers* $\mathbb{N}$ *is the smallest subset of* $\mathbb{N}$ *set such that:*

- 0 *is even,*
- *if* n *is even, then* $S(S(n))$ *is even.*

The language of propositional calculus is the smallest set of expressions such that:

- *If* X *is a propositional variable, then* X *is a propositional expression,*
- $\bot$ *is a propositional expression,*
- *if* A *and* B *are propositional expressions, then* $A \vee B$, $A \wedge B$ *and* $A \Rightarrow B$ *are propositional expressions.*

In typed functional languages like ML but also Rocq, the definition corresponding to the first exemple will be that the type `nat` is the smallest type built with the two *constructors* 0 (or `O`) and `S`.

In OCaml:

```
type nat = O | S of nat;;
```

In Rocq:

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

(there exist syntactical syntactical variants to the latter).

Throughout the course, we will make a heavy use of inductive definitions. Either in set theory when we define the formalisms, or in type theory itself.

We thus take some time to describe the mechanism in set theory. The key which allows all the definitions above to be sound, is that there is a notion of *monotonicity* at play. This appears in the next section: the existence of fix-points is proved for *increasing* functions. We briefly go back to this in 1.3.6.

1.1 Fix-points in lattices

We consider a set $\mathcal{L}$ equipped by an order relation $\leq$.

Definition 1.1.1 (Complete lattice)**.** We say that $(\mathcal{L}, \leq)$ is a **complete join-semilattice** (fr: *sup-demi-treillis*) iff for any $A \subseteq \mathcal{L}$ there exists $\bigvee A \in \mathcal{L}$ which is a least upper bound for A. That is:

(1) $\forall x \in A, x \leq \bigvee A$
(2) $\forall x \in \mathcal{L}, (\forall y \in A, y \leq x) \Rightarrow \bigvee A \leq x.$

Symmetrically, we say that $(\mathcal{L}, \leq)$ is a **complete meet-semilattice** (fr: *inf-demi-treillis*) iff for any $A \subseteq \mathcal{L}$ there exists $\bigwedge A \in \mathcal{L}$ which is a greatest lower bound for A. That is:

(1) $\forall x \in A, \bigwedge A \leq x$
(2) $\forall x \in \mathcal{L}, (\forall y \in A, x \leq y) \Rightarrow x \leq \bigwedge A.$

If $(\mathcal{L}, \leq)$ is both a complete join-semilattice and a complete meet-semilattice, then it is a **complete lattice**.

Lemma 1.1.1. *If $(\mathcal{L}, \leq)$ is a complete join-semilattice (resp. complete meet-semilattice) it bears a maximal element $\top$ (resp. a smallest element $\bot$).*

Proof. Take $\top \equiv \bigvee \mathcal{L}$ (resp. $\bot \equiv \bigwedge \emptyset$). □

Theorem 1.1.2 (Knaster-Tarski)**.** *Let f be a monotonically increasing function:*

$$\forall x, y \in \mathcal{L}, x \leq y \Rightarrow f(x) \leq f(y).$$

If $(\mathcal{L}, \leq)$ is a complete meet-semilattice (resp. join-semilattice) then f has a least fixpoint (resp. a greatest fixpoint).

Proof. Let ut treat the case of the meet-semilattice. Let:

$$A \equiv \{x \in \mathcal{L}, f(x) \leq x\}$$

and:

$$u \equiv \bigwedge A.$$

We notice that A is closed by f:

$$\begin{array}{rcl} x \in A & \iff & f(x) \leq x \\ & \Rightarrow & f(f(x)) \leq f(x) \\ & \Rightarrow & f(x) \in A \end{array}$$

Let z be any element of A. We know that $u \leq z$ and thus $f(u) \leq f(z)$. Since $f(z) \leq z$ this entails $f(u) \leq z$. We thus have shown that $f(u)$ is a lower bound of A.

Since u is the greatest lower bound, this entails that $f(u) \leq u$.

This means that $u \in A$ which in turn implies $f(u) \in A$. Thus $u \leq f(u)$. By antisymmetry we can conclude $u = f(u)$ which means that u is a fixpoint.

Since any fixpoint v is also an element of A, we know that $u \leq v$, so u is indeed the least fixpoint.

We leave the symmetrical case of the join-semilattice as an exercise. □

1.2 Fixpoints of set operators

In practice, we will use the Knaster-Traski theorem for functions operating over sets. Indeed, given any set $\mathcal{U}$, the powerset $\mathcal{P}(U)$ is a complete lattice for the inclusion ordering:

If $A \subseteq \mathcal{P}$ then $\bigcup A$ (resp. $\bigcap A$) is a least upper bound (resp. greatest lower bound) of A.

We can thus look at the least fixpoints of some operators over sets.

Consider the following mapping over sets (actually over sets of expressions):

$$F(X) \equiv \{0\} \cup \{S(x), x \in X\}$$

It is easy to see that F is increasing with respect to inclusion. The least fixpoint of F is thus the set:

$$N \equiv \bigcap \{X, F(X) \subseteq X\}$$

This means that $n \in N$ iff

$$\forall X, F(X) \subseteq X \Rightarrow n \in X$$

that is:

$$\forall X, 0 \in X \vee (\forall x \in X, S(x) \in X) \Rightarrow n \in X$$

We see we naturally retrieve the induction principle over natural numbers; the inductively defined set is precisely:

- the set of objects verifying the induction principle,
- which is another way to put it is the smallest set containing 0 and closed by S.

1.3 Some inductively defined sets and the associated principles

1.3.1 booleans

The set of booleans can be defined as the smallest set containing true and false. Since the definition is not recursive, one does actually not need the fixpoint theorem machinery here. But is may be worth noting that the same unfolding of the fixpoint construction as before gives us the scheme to reason by case over booleans:

$$b \in \mathsf{bool} \iff \forall X, \mathsf{true} \in X \vee \mathsf{false} \in X \Rightarrow b \in X$$

1.3.2 Even numbers

The inductive description of the (sub)set of even natural numbers yields the corresponding induction scheme:

$$n \in \mathsf{Even} \iff \forall X, 0 \in X \wedge (\forall x \in X, S(S(x)) \in X) \Rightarrow n \in X$$

1.3.3 Transitive closures

Consider the set Λ of pure (untyped) λ-terms. We write $t \rhd u$ to state that t rewrites to u in one β-reduction

We want to define the relations $\rhd^+$ and $\rhd^*$ which are respectively the transitive and the transitive-reflexive closures of $\rhd$.

These are typical inductive definitions. The first one can be defined by the clauses:

If $t \rhd u$ then $t \rhd^+ u$,

if $t \rhd^+ u$ and $u \rhd u'$, then $t \rhd^+ u'$.

Note that there are other possible equivalent definitions, but in this one, the left-hand t parameter is the same throughout the definition. This will turn out to be practical and of good taste later on.

The first second can be defined by the clauses:

$t \rhd^* t$,

if $t \rhd^* u$ and $u \rhd u'$, then $t \rhd^* u'$.

Note that there are other possible equivalent definitions, but in the ones chose here, the left-hand t parameter is the same throughout each definitions. This will turn out to be practical and of good taste later on.

Exercise. Write the induction principles associated with these definitions.

1.3.4 A well-foundedness example

Well-foundedness is nicely defined inductively. Typically, it will yield a definition with no "base case". For instance, we can define the set SN of strongly normalizing

λ-terms as the smallest set verifying:

For any λ-term t, if $\forall u \in \Lambda, t \rhd u \Rightarrow u \in \mathsf{SN}$ then $t \in \mathsf{SN}$.

To get a good grip of this example, you can check first that all normal λ-terms are in SN, then that all terms whose direct reducts are normal are in SN, etc.

Exercise. Give a similar definition of the set WN of weakly normalizing λ-terms.

1.3.5 A transfinite example

One can view the least fixpoint obtained through the Knaster-Tarski theorem as the limit of a sequence:

- One has $\emptyset \subseteq F(\emptyset)$,
- thus also $F(\emptyset) \subseteq F(F(\emptyset)), \ldots F^{(i)}(\emptyset) \subseteq F^{(i+1)}(\emptyset) \ldots$.

In many case (and actually in all examples above), the least fixpoint is the limit of this sequence; that is the set $\bigcap_{i \in \mathbb{N}} F^{(i)}(\emptyset)$.

In some cases however, this set is not a fixpoint of F. Then, the fixpoint is reached by $F^{\lambda}(\emptyset)$ where λ is some ordinal larger than ω.

One such definition is the following set of infinitively branching trees, which actually correspond to a set of ordinals:

- $0 \in \mathsf{ord}$,
- if $x \in \mathsf{ord}$, then $S(x) \in \mathsf{ord}$,
- if $\forall i \in \mathbb{N}, u_i \in \mathsf{ord}$, then $\lim((u_i)_{i \in \mathbb{N}}) \in \mathsf{ord}$.

Note however that the construction by the fixpoint through Knaster-Tarski is still correct and does not involve ordinals.

Indeed, the definition of the set ord yields the following induction scheme: $\lambda \in \mathsf{ord}$ if and only if:

$$\begin{array}{l} \forall X. 0 \in X \\ \quad \wedge (\forall x \in X, S(X) \in X) \\ \quad \wedge (\forall (u_i)_{i \in \mathbb{N}} \in X^{\mathbb{N}}. f((u_i)_{i \in \mathbb{N}}) \in X \Rightarrow \mathsf{lim}(f) \in X) \\ \Rightarrow \lambda \in X \end{array}$$

This is a scheme corresponding to a form of transfinite induction.

1.3.6 An unsound definition

Consider the two ML types:

```
type nat = Z | S of nat

type foo = Foo | Loo of (foo -> foo)
```

The first one corresponds to the inductive definition of natural numbers, and the type `nat` can be viewed as inductively defined; meaning it is the least fix-point of an increasing function over terms, as explained at the beginning of this section.

The second one, on the other hand, while accepted by ML, is not inductive. That is because one of the occurrences of `foo` in the type of `Loo` is in a *negative* position. Consequently, the function over sets of terms corresponding to this type definition is not increasing and the Knaster-Tarsky theorem does not apply.

One way to materialize this is to construct a non-terminating program, without using a recursive definition (no `let rec`):

```
let loop x = match x with
  | Loo f -> f (Loo f)
  | _ -> x;;
```

The evaluation of the program `loop (Loo loop)` then loops forever. One can also reconstruct the `let rec` operator from this type.

In other words, there is no induction principle for this type.

1.4 Co-inductive definitions

Inductive definitions are about the smallest set verifying some closure conditions. For some issues however, one may want to use the greatest set verifying the closure conditions. From the construction of the set through the Knaster-Tarski theorem, this corresponds to using the greatest fixpoint instead of the least fixpoint.

This scheme is called co-inductive definitions, as it is dual to the inductive scheme. We do not detail it for now. Let us just mention a few points:

- Co-inductive definitions are a mean to construct infinite objects. For instance the coinductive version of lists corresponds to streams; that is it is possible to construct infinite lists.
- Co-inductive properties are very handy for some applications. Typically, dealing with state transition systems; we want to state that two systems behave the same way, that is they yield the same (infinite) stream of transitions. This relation is called *bisimulation* and is coinductive.
- Rocq's type theory features primitive coinductive definitions alongside its primitive inductive definitions.

Chapter 2

Normalization of Simply Typed λ-Calculi

Lambda terms and λ-calculi will be used extensively in this course. Note that they were invented by Alonzo Church precisely to design a logical formalism (Higher-Order Logic, also called HOL or, originally, the Simple Theory of Types). Since, this tool has found many uses in Logic and Computer Science, and we will use various forms of λ-terms to represent:

- Proofs,
- objects of the formalism, which can be actual functional programs in some formalisms,
- Propositions, or statements in some formalisms.

This will encompass he definition and study of various Types Systems. The basic properties of untyped λ-terms are not in the scope of this course, but we start with some brief definitions and notations. There is a vast literature on this topic. The most well-known ones may be Barendregt [2] and Hindley and Seldin [24].

2.1 Pure λ-Calculus

The aim of λ-calculus is to capture the basic mechanisms of function definition and application. It is a very concise formal language consisting of variables, function application and function abstraction.

Definition 2.1.1 (λ-terms)**.** The set of λ-terms is defined by the following grammar:

$$t \quad ::= \quad x \mid (t\ t) \mid \lambda x.t$$

where x ranges over a given, countable, set of *variables*.

The term $\lambda x.t$ stands for the function mapping x to the (value of the) term t (often written $x \mapsto t$ in informal mathematics).

The term $(t\ u)$ stands for the application of the argument u to the function t, which is genrally written $t(u)$ in informal mathematics. The difference in bracketing is justified by the following notation:

Notation 1. *Given a term t and a sequence of terms $u_1, u_2, \ldots, u_n$, one write $(t\ u_1\ u_2 \ldots\ u_n)$ for $(\ldots((t\ u_1)\ u_2)\ldots)$.*

Definition 2.1.2 (Free Variables)**.** Given a term t, we write $\mathrm{FV}(t)$ for the set of free variables of t, which is defined by the following equations:

$$\begin{aligned} \mathrm{FV}(x) &\equiv \{x\} \\ \mathrm{FV}(t\ u) &\equiv \mathrm{FV}(t) \cup \mathrm{FV}(u) \\ \mathrm{FV}(\lambda x.t) &\equiv \mathrm{FV}(t) \setminus \{x\}. \end{aligned}$$

Definition 2.1.3 (Substitution)**.** Given two terms t and u and a variable x, one defines the term $t[x \setminus u]$ by:

$$\begin{aligned} x[x \setminus u] &\equiv u \\ y[x \setminus u] &\equiv x \quad \text{if } x \neq y \\ (t\ v)[x \setminus u] &\equiv (t[x \setminus u]\ v[x \setminus u]) \\ (\lambda x.t)[x \setminus u] &\equiv \lambda x.t \\ (\lambda y.t)[x \setminus u] &\equiv \lambda y.t[x \setminus u] \quad \text{if } y \notin \mathrm{FV}(u). \end{aligned}$$

When $y \in \mathrm{FV}(u)$, one first *renames* the variable y in the substituted term, chosing a *fresh* variable z:

$$(\lambda y.t)[x \setminus u] \equiv (\lambda z.t[y \setminus z])[x \setminus u] \quad \text{where } z \notin \mathrm{FV}(t) \cup \mathrm{FV}(u)$$

Definition 2.1.4 (α-conversion)**.** More precisely, when $y \notin \mathrm{FV}(t)$, one says that $\lambda x.t$ and $\lambda y.t[x \setminus y]$ are α-convertible, written $\lambda x.t =_\alpha \lambda y.t[x \setminus y]$.

In general, λ-terms are always viewed modulo α-conversion. A precise treatment of variable renaming induces a certain number of technical difficulties. For instance, in the above, substitution and α-conversion should be defined simultaneously. One possible way, especially when doing mechanically checked formal developments, is to use *De Bruijn indices*, where bound variables are represented by the number of binders (or λ-abstractions), separating the variable occurence from its binder. For instance $\lambda x.(x\ \lambda y.(y\ x))$ is then represented by $\lambda.(0\ \lambda.(0\ 1))$. This allows a canonical representation for a λ-term and dispenses with α-conversion. The price to pay is the definition of a number of *lifting* operators, which adjust the indices when performing operations like substitution. We do not detail this and refer to the literature.

In general, in these notes, we use named representation, leaving it open whether the formal representation relies on De Bruijn indices.

The crucial operation in λ-calculus is called β-reduction.

Definition 2.1.5 (β-reduction)**.** A term of the form $(\lambda x.t\ u)$ is called a β-redex. It β-reduces to $t[x \setminus u]$; we write:

$$(\lambda x.t\ u) \to_\beta [x \setminus u].$$

In general, the β-reduction relation, written $\rhd_\beta$, is the contextual closure of $\rightarrow_\beta$.

We write $\rhd_\beta^+$ for the transitive closure of $\rhd_\beta$ (a sequence of one or more reductions), $\rhd_\beta^*$ for the reflexive-transitive closure of $\rhd_\beta$ (a sequence of zero or more reductions) and $=_\beta$ for the reflexive-symmetric-transitive closure. The latter is an equivalence relation and is called β-conversion.

Definition 2.1.6 (Normal terms)**.** A term t is said to be normal when there is no term t' such that $t \rhd_\beta t'$ (or equivalently when no subterm of t is a redex).

When $t \rhd_\beta^* t'$ and t' is normal, t' is said to be a *normal form* of t.

Lemma 2.1.1 (Characterization of normal terms)**.** *Any normal λ-term is of the form*

$$\lambda x_1.\lambda x_2.\ldots\lambda x_n.(y\ u_1\ \ldots\ u_m)$$

where $u_1, \ldots, u_m$ are normal terms.

Some terms have no normal form. The most well-known one is $\lambda x.(x\ x)\ \lambda x.(x\ x)$ which reduces to itself. An important variant, given a term f is $Y_f \equiv \lambda x.f\ (x\ x)\ \lambda x.f(x\ x)$ since one has:

$$Y_f \rhd_\beta f\ Y_f$$

that is Y_f is the fix-point combinator of f. The existence of this combinator is key for showing that the λ-calculus is Turing complete.

Definition 2.1.7 (Church numerals)**.** Given a natural number n, the Church encoding of n is the λ-term which iterates a function n times over an argument. That is, if we call $\overline{n}$ the Church numeral for n:

$$\begin{aligned}
\overline{0} &\equiv \lambda f.\lambda x.x \\
\overline{1} &\equiv \lambda f.\lambda x.(f\ x) \\
\overline{2} &\equiv \lambda f.\lambda x.(f\ (f\ x)) \\
&\ldots \\
\overline{n} &\equiv \lambda f.\lambda x.(f^{(n)}\ x)
\end{aligned}$$

On can also encode the ordered pair (a, b) by the function taking (a function) f as an argument and applying it to a and b:

$$(a, b) \equiv \lambda f.(f\ a\ b).$$

Then the two projections can be defined as:

$$\pi_1\ c \equiv c\ \lambda a.\lambda b.a \qquad \pi_2\ c \equiv c\ \lambda a.\lambda b.b$$

and one can then check that $\pi_1\ (a, b) \rhd_\beta^* a$ and $\pi_2\ (a, b) \rhd_\beta^* b$.

Theorem 2.1.2 (Church-Rosser)**.** *Given two terms t and u, if $t =_\beta u$, then there exists a term v such that $t \rhd_\beta^* v$ and $u \rhd_\beta^* v$.*

2.2 Types and normalization proofs

From now on, we will mainly restrict ourselves to *typed* λ-terms. That is:

1. We will define various type systems, through an inductively defined judgement $\vdash t : T$ or $\Gamma \vdash t : T$. Each time, the typing derivation will closely follow the structure of the term t.
2. We will then prove normalization for the type system: if $t : T$ (resp. $\Gamma \vdash t : T$) then $t \in \mathsf{SN}$. This will be proved by induction over t or over the typing derivation, both formulations being equivalent.

This will be done for a collection of, increasignly powerful, type systems. In this chapter we only consider simply typed λ-calculus, then extend it with sums and pairs, and then consider Gödel's System T, which allows structural recursion over natural numbers.

Every time, the main difficulty will be to formulate the correct induction hypothesis. One can check that 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.

2.3 Simply typed lambda-calculus

Before turning to the reducibility proof, let us first recall the (Curry-style) typing system for simply typed λ-calculus restricted to the arrow type.

Types and (raw, unannotated) terms are given by the grammars:

$$T \quad ::= \quad A \ \mid \ T \to T \qquad\qquad t, u \quad ::= \quad x^T \ \mid \ \lambda x.t \ \mid \ (t\ u)$$

where A ranges over atomic types.

The typing judgement $\vdash t : T$ is defined by the following rules:

$$\frac{}{\Gamma \vdash x^T : T}\ (\textsc{Var}) \qquad \frac{\vdash t : U}{\vdash \lambda x^T.t : T \to U}\ (\textsc{Abs})$$

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

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

Definition 2.3.1. For any type T we define a set $|T|$ of λ-terms by:

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

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 $t \in \mathsf{SN}$, which means that we have indeed strengthened the induction hypothesis.

If the plan is simple, the details will be more intricate.

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

Definition 2.3.2 (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 2.3.1. *For every type T the three following properties hold:*

1. $|T| \subset \mathsf{SN}$.
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 conjunction of the three assertions is proved by induction over T.

If T is atomic then $|T| = \mathsf{SN}$. 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 then 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), $|V| \subset \mathsf{SN}$, so $(t\ x) \in \mathsf{SN}$ and thus $t \in \mathsf{SN}$.

(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 $u_0 \in \mathsf{SN}$. 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 $(\lambda x.t\ u_0\ u_1\ \ldots u_n\ u) \in |V|$. □

Any normalization proofs needs to take substitution into account. In typed calculi, variables are only substituted by terms of their type (that is x^T can only be substituted by terms of type T). This leads to the following definition:

Definition 2.3.3 (correct valuation)**.** We call valuation a function $\mathcal{I}$ which associates a λ-term to each variable. We say that this valuation is correct when for every variable x^T we have:

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

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

We can state and prove the main lemma which is the *raison d'être* of all the definitions:

Lemma 2.3.2. *If $\vdash t : T$, then for any correct valuation $\mathcal{I}$, we have: $|t|_{\mathcal{I}} \in |T|$.*

Proof. We reason by induction over t.

- If t is of the form x^T, then $|x^T|_{\mathcal{I}} = \mathcal{I}(x^T)$ which belongs to $|T|$ because $\mathcal{I}$ is correct.
- If t is of the form $(u\ v)$, we know that $u : V \to T$ and $v : V$. So $|u|_{\mathcal{I}} \in |V \to T|$ and $|v|_{\mathcal{I}} \in |V|$. Thus $|(u\ v)|_{\mathcal{I}} = (|u|_{\mathcal{I}}\ |v|_{\mathcal{I}}) \in |T|$.
- 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, $|u|_{\mathcal{J}} \in |U|$ for any adapted $\mathcal{J}$.

  Given $v \in |V|$ and some adapted $\mathcal{I}$, we need to show that $(|\lambda x^V.u|_{\mathcal{I}}\ v) \in |U|$.

  We have: $|\lambda x^V.u|_{\mathcal{I}} = \lambda x^V.|u|_{\mathcal{I};x^V \leftarrow x^V}$. We must thus prove

  $$(\lambda x^V.|u|_{\mathcal{I};x^V \leftarrow x^V}\ v) \in |U|.$$

  Because of the third property of lemma 2.3.1, it is sufficient to show

  $$|u|_{\mathcal{I};x^V \leftarrow x^V}[x \setminus v] \in |U|.$$

  Because substitution avoids variable capture, we can consider that x is not free in $\mathcal{I}$ and thus $|u|_{\mathcal{I};x^V \leftarrow x^V}[x \setminus v] = |u|_{\mathcal{I};x^V \leftarrow v}$.

  It is easy to see that $\mathcal{I}; x^V \leftarrow v$ is adapted, and thus we have indeed $|u|_{\mathcal{I};x^V \leftarrow v} \in |U|$.

□

2.4 Proof variants

There are several variants of the proof above. In particular, there are alternative formulations to the lemma 2.3.1 which are similar but not exactly equivalent. A quite elegant one is the one of Girard [21].

Definition 2.4.1 (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 2.4.1. *For any type T, the set $|T|$ verifies the following conditions:*

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

(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 2.3.2, with little technical differences.

Definition 2.4.2. 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 $\mu(t) = 0$ and when $t \rhd_\beta t'$ then $\mu(t') < \mu(t)$.

Lemma 2.4.2. *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.

□

We can now give the new proof of lemma 2.3.2 itself:

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

By definition of $|V \to U|$, this means proving that for any $v \in |V|$:

$$(\lambda x^V.|u|_{\mathcal{I};x^V \leftarrow x^V} v) \in |U|.$$

We will do this by proving by induction over $\mu(|u|_{\mathcal{I};x^V \leftarrow x^V})+\mu(v)$ that all reducts of $(\lambda x^V.|u|_{\mathcal{I};x^V \leftarrow x^V} v)$ are in $|U|$.

The term $(\lambda x^V.|u|_{\mathcal{I};x^V \leftarrow x^V}\ v)$can reduce to:

- $(\lambda x^V.|u|_{\mathcal{I};x^V \leftarrow x^V}\ v')$ where $v \rhd v'$. But this is in $|V|$ since $\mu(v') < \mu(v)$.
- $(\lambda x^V u' v)$ where $|u|_{\mathcal{I};x^V \leftarrow x^V} \rhd u')$. But this is in $|V|$ since $\mu(u') < \mu(|u|_{\mathcal{I};x^V \leftarrow x^V})$.
- $|u|_{\mathcal{I};x^V \leftarrow x^V})[x^V \backslash v]$ which, by the properties of substitution is equal to $|u|_{\mathcal{I};x^V \leftarrow v}$. Because $\mathcal{I}; x^V \leftarrow v$ is a correct interpretation, this is indeed a consequence of the general (outermost) induction ensures that $|u|_{\mathcal{I};x^V \leftarrow v} \in |U|$.

□

2.5 Product and sum types

A first extension of simply typed λ-calculus is the addition of product and sum types.

2.5.1 The system

The algebra of types is now generated by the following grammar:

$$T \quad ::= \quad A \mid T \to T \mid T \times T \mid T + T \qquad \text{where } A \text{ stands for an atomic type}$$

The terms themselves are enriched with pairs and projections (for product types) and constructors and a simple pattern-matching (for sum types).

$$t \quad ::= \quad x^T \mid \lambda x^T.t \mid (t\ t) \mid (t,t) \mid \pi_1(t) \mid \pi_2(t) \mid i(t) \mid j(t) \mid \delta(t, x^T.t, x^T.t)$$

The β-reduction is extended by:

$$\begin{array}{rcl} \pi_1(t_1,t_2) & \rhd & t_1 \\ \pi_2(t_1,t_2) & \rhd & t_2 \\ \delta(i(t), x^U.u, y^V.v) & \rhd & u[x^U \setminus t] \\ \delta(j(t), x^U.u, y^V.v) & \rhd & v[y^V \setminus t] \end{array}$$

Finally, the set of typing rules is extended by:

$$\frac{\vdash t : T \quad \vdash u : U}{\vdash (t,u) : T \times U}$$

$$\frac{\vdash t : U \times V}{\vdash \pi_1(t) : U} \qquad \frac{\vdash t : U \times V}{\vdash \pi_2(t) : V}$$

$$\frac{\vdash t : U}{\vdash i(t) : U + V} \qquad \frac{\vdash t : V}{\vdash j(t) : U + V}$$

$$\frac{\vdash t : U + V \quad \vdash u : T \quad \vdash v : T}{\delta(t, x^U.u, y^V.v) : T}$$

The basic metatheoretic lemmas are similar to the fragment built solely with $\to$.

Lemma 2.5.1. *In this calculus:*

- $\vdash (u,v) : T$ *if and only if* T *is of the form* $U \times V$ *with* $\vdash u : U$ *and* $\vdash v : V$,
- $\vdash \pi_1(t) : U$ *if and only if* $\vdash t : U \times V$ *for some* V,
- $\vdash \pi_2(t) : V$ *if and only if* $\vdash t : U \times V$ *for some* U,
- $\vdash i(t) : T$ *if and only if* T *is of the form* $U + V$ *and* $\vdash t : U$,
- $\vdash j(t) : T$ *if and only if* T *is of the form* $U + V$ *and* $\vdash t : V$,
- $\vdash \delta(t, x^U.u, y^V.v) : T$ *if and only is* $\vdash t : U + V$, $\vdash u : T$ *and* $\vdash v : T$.

Lemma 2.5.2. *If* $\vdash t : T$ *and* $\vdash u : U$ *then* $\vdash t[x^U \setminus u] : T$.

Lemma 2.5.3. *If* $\vdash t : T$ *and* $t \rhd t'$ *then* $\vdash t' : T$.

The following lemma is simple, but will appear often in this course.

Lemma 2.5.4. *Any closed normal term of a type $T \to U$ is of the form $\lambda x^T.u$. Any closed normal term of type $T \times U$ is of the form (t, u) (where t and u are themselves closed normal terms of respective types T and U). Any closed normal term of type $T + U$ is either of the form $i(t)$ or of the form $j(u)$ with t (resp. u) being a closed normal term of type T (resp. U).*

2.5.2 Reducibility

The reducibility proof is very similar to the previous one, even if one has to take care of more cases. To be precise, it is easier to extend the version of section 2.4.

One possible definition of reducibility is:

Definition 2.5.1. We extend the definition 2.3.1 by the following clauses:

- $|U \times V| \equiv \{t \in \mathsf{SN}, t \rhd^* (u, v) \Rightarrow u \in |U| \wedge v \in |V|\}$
- $|U + V| \equiv \{t \in \mathsf{SN}, (t \rhd^* i(u) \Rightarrow u \in |U|) \wedge (t \rhd^* j(v) \Rightarrow v \in |V|)\}$

We then have to extend the definition of neutral terms.

Definition 2.5.2. The set of $\mathcal{N}$ of neutral terms is the set of terms which are not of one of the following forms: $\lambda x^T.t$, (t, u), $i(t)$, $j(t)$.

The next lemma is restated unchanged, but for the definition of $\mathcal{N}$.

Lemma 2.5.5. *For every type T the three following properties hold:*

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

Proof. The new cases in the proof are quite similar to the previous ones. Here are some.

Proving (2) when $T = U \times V$. If $t \in |U \times V|$ and $t \rhd t'$. Then we are sure that $t' \in \mathsf{SN}$ since $t \in \mathsf{SN}$. One the other hand, if $t' \rhd^* (u, v)$, then $t \rhd^* (u, v)$. So we know that $u \in |U|$ and $v \in |V|$.

Proving (3) when $T = U \times V$. If all reducts of t are in $|U \times V|$, then they are all SN, thus $t \in \mathsf{SN}$.

Furthermore, if $t \rhd^* (u, v)$ and t is neutral, there is at least one reduction taking place: $t \rhd t' \rhd^* (u, v)$. Thus, $t' \in |U \times V|$, which ensures that $u \in |U|$ and $v \in |V|$.

Proving (3) when $T = U + V$. Again, if all reducts of t are in $|U + V|$, then t is in SN.

Furthermore, if $t \in \mathcal{N}$ and $t \rhd^* i(u)$, then there exists t' such that $t \rhd t' \rhd^* i(u)$. If $t' \in |U + V|$, we then know that $u \in |U|$. □

The main lemma is also unchanged in its phrasing.

Lemma 2.5.6. *If $\vdash t : T$, then for any correct valuation $\mathcal{I}$, we have: $|t|_{\mathcal{I}} \in |T|$.*

Proof. We only detail some cases which have not been treated in the previous proof.

- If t is of the form (u, v) with $T = U \times V$, $\vdash u : U$ and $\vdash v : V$, we have $|u|_{\mathcal{I}} \in |U|$ and $|v|_{\mathcal{I}} \in |V|$. We have $|(u, v)|_{\mathcal{I}} = (|u|_{\mathcal{I}}, |v|_{\mathcal{I}})$. So any reduct of $|(u, v)|_{\mathcal{I}}$ is of the form (u', v') with $|u|_{\mathcal{I}} \rhd^* u'$ and $|v|_{\mathcal{I}} \rhd^* v'$. So $u' \in |U|$ and $v' \in |V|$, which is sufficient to ensure that $|(u, v)|_{\mathcal{I}} \in |U \times V|$.
- If t is of the form $\pi_1(w)$, then $|t|_{\mathcal{I}} = \pi_1(|w|_{\mathcal{I}})$. We know that $\vdash w : T \times U$ and thus $|w|_{\mathcal{I}} \in |T \times U|$. This implies that $|w|_{\mathcal{I}} \in \mathsf{SN}$. Since $\pi_1(w) \in \mathcal{N}$, it is sufficient for $\pi_1(|w|_{\mathcal{I}}) \in |T|$ to check that if $\pi_1(|w|_{\mathcal{I}}) \rhd t'$ then $t' \in |T|$. We can reason by induction over $\mu(|w|_{\mathcal{I}})$. If $\pi_1(|w|_{\mathcal{I}}) \rhd t'$, then t' can be:
  - $\pi_1(w')$ with $|w|_{\mathcal{I}} \rhd w'$ which is taken care of by the inner induction.
  - v if $|w|_{\mathcal{I}}$ is of the form (v, u). But then $|w|_{\mathcal{I}} \rhd^* (v, u)$ and thus $v \in |T|$.

□

We also extend the type system with a type unit, with a single canonical inhabitant $()$ and no reduction rule of its own: this addition is trivially compatible with strong normalization, since it introduces no new redex.

Corollary 2.5.7 (Strong Normalization)**.** *Every term typable in the simply typed λ-calculus extended with product, sum and* unit *types is strongly normalizing.*

Proof. By Theorem 2.5.5 (property 1), $|T| \subset \mathsf{SN}$ for every type T; by Theorem 2.5.6, every term typable in context belongs to the reducibility candidate of its type (take $\mathcal{I}$ to be the identity valuation on the free variables), hence is in SN. Adding unit changes nothing to this argument since it carries no reduction rule. □

2.6 System T

Gödel's System T was introduced in 1958 in order to study cut elimination in arithmetic - which we will do in chapter **??**. We here study it on one hand because it allows to prove cut elimination for artihmetic and later 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 Rocq.

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 $0 : N$,
- a constant $S : N \to N$,

- for every type T, a primitive construct $R_T(t_0, t_S, t)$, coming with an additional, dedicated, typing rule:

$$\frac{\vdash t_0 : T \vdash t_S : N \to T \to T \vdash t : N}{\vdash R(t_0, t_S, t) : T}$$

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

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

Note that we chose R_T is *not* to be a curried function of type $N \to T \to (N \to T \to T) \to T$ applied one argument at a time: it takes its three arguments at once. This is a purely technical matter, it could be done the other way, but the definition above will make the SM proof smoother.

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

The definition of reducibility is similar to simply typed calculus.

Definition 2.6.1. For every typed T, the set of reducible terms is defined by:

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

Definition 2.6.2. The set $\mathcal{N}$ of neutral terms is the set of terms which are not of the forms $\lambda x^V.u, 0, (S\ v)$.

Lemma 2.6.1. *For every type T, the following propositions hold:*

1. $|T| \subset \mathsf{SN}$
2. $\forall t \in |T|, t \rhd t' \Rightarrow t' \in |T|$
3. $\forall t \in \mathcal{N}, (\forall t', t \rhd t' \Rightarrow t' \in |T|) \Rightarrow t \in |T|$

Proof. The Proof is similar to the one for simply-typed λ-calculus. We proceed again by induction over the strucure of T.

The case of atomic terms follows the same simple argument than for simple types.

If $T = U \to V$.

1. The induction hypothesis (3) for U ensures that any variable x is an element of $|U|$, so if $t \in |U \to V|$, then $(t\ x) \in |V|$, which implies that $(t\ x) \in \mathsf{SN}$, so $t \in \mathsf{SN}$.
2. Take $t \in |U \to V|$ and $u \in |U|$. We know $(t\ u) \in |V|$, so the induction hypothesis (2) for V ensures that if $t \rhd_\beta t'$, then $(t'\ u) \in |V|$ and so $t' \in |U \to V|$.

3. Consider $t \in \mathcal{N}$, such that if $t \rhd_\beta t'$ then $t' \in |U \to V|$. Given $u \in |U|$, we need to show that $(t\ u) \in |V|$. Using hypothesis (3) for V, it is sufficient to show that if $(t\ u) \rhd_\beta v$, then $v \in |V|$.

   We can show this by induction over $\mu(u)$. Because t is neutral, $(t\ u)$ cannot be a redex, so if $(t\ u) \rhd_\beta v$, either:

   - $v = (t'\ u)$ with $t \rhd_\beta t'$, in which case know that $t' \in |U \to V|$ and thus $(t'\ u) \in \|V|$.
   - $v = (t\ u')$ with $u \rhd_\beta u'$, in which case $\mu(u') < \mu(u)$, so $(t\ u') \in |V|$.

□

The main lemma is stated as before, except that we now consider the extended type system:

Lemma 2.6.2. *If $\vdash t : T$ in system T, then for any correct valuation $\mathcal{I}$, we have: $|t|_\mathcal{I} \in |T|$.*

Proof. Again, we reason by induction ov the term t, or equivalently over the typing derivation.

$t = x^T$ Then $|t|_\mathcal{I} = \mathcal{I}(x^T)$ which is in $|T|$ because $\mathcal{I}$ is correct.

$t = (u\ v)$ Similar to the proof for simply typed calculus.

$t = \lambda x^U.v$ Similar to the proof for simply typed calculus.

$t = 0$ It is obvious that $|0|_\mathcal{I} = 0 \in \mathsf{SN}$.

$t = S(u)$ Typing ensures that $\vdash u : N$, so by induction hypohtesis, $|u|_\mathcal{I} \in |N| = \mathsf{SN}$. Thus $|S(u)|_\mathcal{I} = S(|u|_\mathcal{I}) \in \mathsf{SN}$.

$t = R_T(t_0, t_S, u)$ Then the induction hypotheses ensure:

- $|u|_\mathcal{I} \in \mathsf{SN}$,
- $|t_0|_\mathcal{I} \in |T|$,
- $|t_S|_\mathcal{I} \in |N \to T \to T|$.

Since $|R_T(t0, t_S, u)|_\mathcal{I}$ is equal to $R_T(|t0|_\mathcal{I}, |t_S|_\mathcal{I}, |u|_\mathcal{I})$ we need to prove the latter term is in $|T|$. Since it is a neutral term, we only need to prove that ny of its reducts is in $|T|$.

We do this by induction over $\mu(|t_0|_\mathcal{I}) + \mu(|t_S|_\mathcal{I}) + \mu(|u|_\mathcal{I}) + \#(|u|_\mathcal{I})$, where $\#(v)$ is the size of the normal form of a normalizable term v.

Or, more precisely, given any

- $v \in \mathsf{SN}$,
- $v_0 \in |T|$,
- $v_S \in |N \to T \to T|$,

we prove by induction over $\mu(v_0)+\mu(v_S)+\mu(v)+\#(v)$, that $R_T(v_0, v_S, v) \in |T|$.

The possible reducts of $R_T(v_0, v_S, v)$ are:

- $R_T(v'_0, v_S, v), R_T(v_0, v'_S, v), R_T(v'_0, v_S, v)$ where either $v_0 \rhd_\beta v'_0$, $v_S \rhd_\beta v'_S$ or $v \rhd_\beta v'$. In all cases the measure has decreased and the resulting term is in $|T|$ by induction hypothesis.
- v_0 (when $v = 0$) which is known to be in $|T|$.
- $(v_S\ p\ R_T(v_0, v_S, p))$ when $v = S(p)$. We then know that:
  - $p \in |N|$,
  - because $\#(p) < \#(v)$ and $\mu(p) = \mu(v)$, the induction hypothesis ensures that $R_T(v_0, v_S, p) \in |T|$.

  Thus, since $v_S \in |N \to T \to T|$, we know that $(v_S\ p\ R_T(v_0, v_S, p)) \in |T|$.

□

Remark. One can combine the system T with product and sum types. In that case, the set of neutral terms is the set of terms which are not of the forms $\lambda x^V.u, 0, (S\ v), (u, v), i(u), j(v)$.

The normalization proofs for system T, sum and product types merge without problem.

Corollary 2.6.3 (Strong Normalization for System T)**.** *Every term typable in System T, extended with product, sum and* unit *types as above, is strongly normalizing.*

Proof. Same argument as Theorem 2.5.7, using Theorem 2.6.1 in place of Theorem 2.5.5. □

2.7 Unit Type

We will sometimes use an additional singleton type; this type and the corresponding constructions can be added to any of the calculi above (simple types with or without sum types, system T). We use the notation from ML, calling this type unit and its single canonical element (). That is we extend the algebra of terms with the following grammar entries:

$$t \quad ::= \quad \ldots \mid () \mid \mathsf{check}\ t.t$$

and β-reduction with the following clause:

$$\mathsf{check}\ ().t \rhd_\beta t.$$

We do not detail how this does not jeopardize Church-Rosser, subject reduction, and other besic properties.

This new reduction does not add any new complexity, and the calculi above extended with the unit type are all strongly normalizable. This can easily be proved in various ways.

Reduciblity for unit type

We can the same reducibility proofs as above taking: $|\mathsf{unit}| = \mathsf{SN}$ and $\mathcal{N}$ is the set of terms which are not of the forms $\lambda x^T.t$, (t, u), $i(t)$, $j(t)$, $()$.

We do not detail the additional steps in the proofs.

Simulating unit type reductions

We can mimick the reductions of the unit type in simply typed λ-calculus, and thus also in all its extensions by sum, product or N types.

Then unit is just a regular atomic type. We define a mapping from λ-terms in the calculus with unit type to its counterpart without unit type by:

$$\begin{array}{rcll} ||x^T|| & \equiv & x^T & \\ ||\lambda x^T.t|| & \equiv & \lambda x^T.||t|| & \\ ||(t\ u)|| & \equiv & (||t||\ ||u||) & \\ ||()|| & \equiv & x^{\mathsf{unit}} & \text{where } x^{\mathsf{unit}} \text{ is an arbitrary chosen variable} \\ ||\mathsf{check}\ t.u|| & \equiv & (\lambda x^N.||u||\ ||t||) & \end{array}$$

It is the straightforward to show:

1. If $\vdash t : T$ (with the unit rules) then $\vdash ||t|| : T$ (in the system without unit rules).
2. If $t \rhd_\beta u$ (including by the unit rule) then $||t|| \rhd_\beta ||u||$ (again without the unit rule).

It follows that if the system without unit rule is strongly normalizable, so is the system with the unit rule.

Chapter 3

First Order Logic

First-Order logic (FOL) is also called *predicate calculus* (in french *logique du premier ordre* and *calcul des prédicats*). It is a framework in which one can define various mathematical languages and for which two main sets of logical deduction rules exist: *sequent calculus* and *natural deduction*; in this course we mainly focus on the latter. Also, even if we are interested by proof systems implementing more expressive languages, first-order logic remains a central fragment of most logical formalisms, and many notions of FOL are still present in modern proof assistants.

We suppose the student or reader has been exposed to first-order logic before, but we here:

1. Give the definitions to set the frame and foundations,

2. insist on the notion of *cuts*, the properties of cut-free proofs and the question of cut elimination, since they will be crucial later on.

3.1 First-Order Languages and Theories

A first-order language is defined by a countably infinite set of variables $\mathcal{X} = \{x, y, z, \ldots\}$, a ranked[1] set $\Sigma = \{f, g, h, \ldots\}$ of *function symbols* and a ranked set $\mathcal{P} = \{P, Q, R, \ldots\}$ of *predicates*.

Definition 3.1.1 (Terms, formulas)**.** The set of *term* $\mathcal{T}$ of the language is inductively defined as follows:

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

The set of formulas $\mathcal{F} = \{A, B, \ldots\}$ of the language is inductively defined as

[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.

follows:

$$\begin{array}{rll} A, B \in \mathcal{F} := & P(t_1, \cdots, t_n) & (P_{/n} \in \mathcal{P}) \\ | & \bot & \text{(false proposition)} \\ | & A \wedge B & \text{(conjunction)} \\ | & A \vee B & \text{(disjunction)} \\ | & A \Rightarrow B & \text{(implication)} \\ | & \forall x.\, A & \text{(universal quantification)} \\ | & \exists x.\, A & \text{(existential quantification)} \end{array}$$

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

A first-order theory is then given by a language and a set of propositions which are assumed to be true:

Definition 3.1.2 (First-order theory)**.** A *first-order theory* is given by a first-order language $(\mathcal{X}, \Sigma, \mathcal{P})$ together with a a set of *closed* formulas called of this language, called the *axioms* of the theory.

3.2 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 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.

A basic idea is that:

- The introduction rule(s) give(s) the *canonical* way(s) to prove a proposition. For instance providing proving $A \wedge B$ by providing a proof of A on one hand, and a proof of B on the other.
- The elimination rule(s) give(s) the canonical way(s) to use a previously proven proposition. For instance, knowing $A \Rightarrow B$, to combine it with a proof of A to obtain a proof of B.

The name natural deduction was chosen by Gentzen because the logical rules are meant to be close to the way actual manthematicians build actual proofs. Although this caracterization is today dated in several respects, it obviously explains why we are interested by natural deduction in a course focusing on the use of formalisms in proof assistants.

Definition 3.2.1 (Natural Deduction Rules)**.** We call *sequent* any pair $\Gamma \vdash A$, where Γ is a finite set of formulas and A is a formula. A sequent $\Gamma \vdash A$ is *provable (in*

Non Structural Rules

$$\frac{A \in \Gamma}{\Gamma \vdash A}\ (\text{Ax}) \qquad \frac{}{\Gamma \vdash A \vee \neg A}\ (\text{EM})$$

Introduction Rules

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ (\vee\text{-I}_1) \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ (\vee\text{-I}_2) \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ (\wedge\text{-I})$$

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

Eliminiation Rules

$$\frac{\Gamma \vdash \mathsf{false}}{\Gamma \vdash A}\ (\mathsf{false}\text{-E}) \qquad \frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash D \qquad \Gamma, B \vdash D}{\Gamma \vdash D}\ (\vee\text{-E})$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ (\wedge\text{-E}_1) \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ (\wedge\text{-E}_2) \qquad \frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ (\Rightarrow\text{-E})$$

$$\frac{\Gamma \vdash \forall x.\, A}{\Gamma \vdash A[x \mapsto t]}\ (\forall\text{-E}) \qquad \frac{\Gamma \vdash \exists x.\, A \qquad \Gamma, A \vdash B \qquad x \notin \text{FV}(\Gamma, B)}{\Gamma \vdash B}\ (\exists\text{-E})$$

Figure 3.1: Natural Deduction Rules

natural deduction) if it can be inductively derived from the rules of Figure 3.1. A *proof (in natural deduction)* of a sequent $\Gamma \vdash A$ is a tree:

- where nodes are labelled by a sequent,
- whose root is labelled by the sequent $\Gamma \vdash A$, and
- s.t. for any node labelled with the sequent $\Delta \vdash B$ and whose children are resp. labelled with sequents $\Delta_i \vdash B_i$ ($i \in \{1..n\}$), there exists a rule of Figure 3.1 of the form:

$$\frac{\Delta_1 \vdash B_1 \quad \cdots \quad \Delta_n \vdash B_n}{\Delta \vdash B}$$

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

Example. Figure 3.2 gives a proof in natural deduction of the proposition $A \wedge B \Rightarrow B \wedge A$.

Definition 3.2.2 (Natural deduction proof in a theory)**.** Let $\mathcal{T}$ be a first-order theory. A formula A is provable in $\mathcal{T}$, written $\mathcal{T} \vdash A$, iff there exists a finite subset Γ of the axioms of $\mathcal{T}$ s.t. $\Gamma \vdash A$.

$$\dfrac{\dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}\,(\text{Ax})}{A \wedge B \vdash B}\,(\wedge\text{-E}_2) \qquad \dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}\,(\text{Ax})}{A \wedge B \vdash A}\,(\wedge\text{-E}_1)}{A \wedge B \vdash B \wedge A}\,(\wedge\text{-I})}{\vdash A \wedge B \Rightarrow B \wedge A}\,(\Rightarrow\text{-I})$$

Figure 3.2: Proof in Natural Deduction of $A \wedge B \Rightarrow B \wedge A$

Definition 3.2.3 (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.

Definition 3.2.4 (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 3.2.1 (Correctness)**.** *If* $\Gamma \vdash A$*, then* $\vDash \bigwedge_{B \in \Gamma} B \Rightarrow A$*. Likewise, if* $\mathcal{T} \vdash A$*, then* $\mathcal{T} \vDash A$*.*

Lemma 3.2.2 (Completeness)**.** *If* $\vDash \bigwedge_{B \in \Gamma} B \Rightarrow A$*, then* $\Gamma \vdash A$*. Likewise, if* $\mathcal{T} \vDash A$*, then there exists a finite subset* Γ *of axioms of* $\mathcal{T}$ *s.t.* $\Gamma \vdash A$*.*

3.3 Two Well Known Theories

3.3.1 Arithmetic

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.

Arithmetic is a remarkable theory for various reasons. For instance, it was defined as early as 1889 by Guiseppe Peano (see [38]). 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:

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

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

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

$$[P] \quad P(0) \wedge (\forall x.\, P(x) \Rightarrow P(S(x))) \Rightarrow \forall x.\, P(x). \tag{7}$$

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:

$$P[x \mapsto 0] \wedge (\forall x.\, P \Rightarrow P[x \mapsto S(x)]) \Rightarrow \forall x.\, P.$$

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:

$$\forall x.\, x = x \tag{8}$$
$$[P] \quad \forall x.\, \forall y.\, P(x) \wedge x = y \Rightarrow P(y). \tag{9}$$

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.

3.3.2 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:

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

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

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

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 together with the axiom $\forall a\ b.\, b \in \mathrm{Union}(a) \iff \exists d.\, b \in d \wedge d \in a$.

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 $\forall x.\, x \in a \Rightarrow x \in b$.

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 [32].

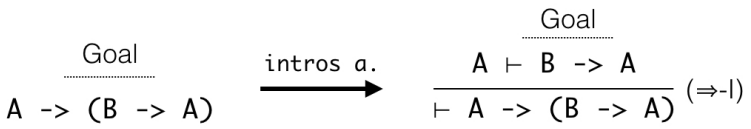
3.4 Formal proof construction

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). One 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 ⇒-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.

3.5 Logical cuts

The questions of cuts and cut-elimination are central in proof theory.

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 3.5.1 (Cuts 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 3.3 gives an extensive listing of possible cuts. A proof that does not contain any cut is said to be *cut-free*.

3.6 Properties of cut-free proofs

Lemma 3.6.1. *An intuitionistic cut-free proof of* $[] \vdash A$ *ends with an introduction rule.*

Proof. By induction over the structure of the proof.

- The proof cannot end with an axiom rule, since there are no hypotheses.
- If the proof ends with the first elimination rule for conjunction:

$$\dfrac{\dfrac{\sigma}{\vdash A \wedge B}}{\vdash A}\ (\wedge\text{-E}_1)$$

  the by induction hypothesis, σ ends with an introduction rule, which has to be $\wedge$-I. So there is a cut, which means this contradicts the initial hypothesis.
- The other elimination rules are all similar. The crucial point is that for all elimination rules, the set of hypotheses of the conclusion is the same as the one of the main premiss. For instance looking at the elimination rule for disjunction, with premisses $\Gamma \vdash A \vee B$, $\Gamma; A \vdash C$ and $\Gamma; B \vdash C$, the induction hypothesis only ensures that the first one ends with an introduction rule. But this is precisely what is needed to prove that a cut-free, axiom-free proof cannot end with the elimination rule.

□

Corollary 3.6.2. *There is no intuitionistic cut-free proof of* $[] \vdash \bot$.

Corollary 3.6.3. *An intuitionistic cut free proof of* $[] \vdash A \vee B$ *either is a proof of* $[] \vdash A$ *followed by* $\vee$-I_1 *or a proof of* $[] \vdash B$ *followed by* $\vee$-I_2.

Corollary 3.6.4. *An intuitionistic cut free proof of* $[] \vdash \exists x.A$ *is a proof of* $[] \vdash A[x \backslash t]$, *for a certain* t, *followed by* $\exists$-I.

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B}\,(\Rightarrow\text{-I}) \qquad \dfrac{\vdots}{\Gamma, A \vdash B}}{\Gamma \vdash B}\,(\Rightarrow\text{-E})$$

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

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

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash A} \qquad \dfrac{\vdots}{\Gamma \vdash B}}{\Gamma \vdash A \wedge B}\,(\wedge\text{-I})}{\Gamma \vdash A}\,(\wedge\text{-E}_1) \qquad \dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash A} \qquad \dfrac{\vdots}{\Gamma \vdash B}}{\Gamma \vdash A \wedge B}\,(\wedge\text{-I})}{\Gamma \vdash B}\,(\wedge\text{-E}_2)$$

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

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

Figure 3.3: Listing of Logical Cuts in Natural Deduction

3.7 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 3.7.1 (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 3.7.2. *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:

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

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…)

3.8 Axiomatic Cuts

For certain axioms, we can define a corresponding notion of cut. This is useful for important axioms of arithmetic.

3.8.1 Equality Cuts

The reflexivity axiom is the canonical way to prove equality, and thus can be viewed as a kind of introduction rule. The Leibniz scheme on the other hand corresponds to a form of elimination. Consider the following situation where both are used one after the other:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \forall x.\, x = x}\ (\text{Ax})}{\vdash t = t}\ (\forall\text{-E}) \quad \dfrac{\sigma}{\vdash P(t)}}{\vdash t = t \wedge P(t)} \quad \dfrac{\dfrac{}{\vdash \forall x. \forall y. x = y \wedge P(x) \Rightarrow P(y)}\ (\text{Ax})}{\vdash t = t \wedge P(t) \Rightarrow P(t)}}{\vdash P(t)}\ (\forall\text{-E } (\times 2))$$

We can view this as an axiomatic cut for equality, which can be simplified to the derivation σ.

3.8.2 Induction Cuts

Induction cuts occur when a property $\forall x.A$ is proved using the induction axiom, and the quantifier is eliminated by instantiating x either by 0 or a successor.

Let us write

$$\dfrac{\dfrac{\sigma_0}{\vdash P(0)} \qquad \dfrac{\sigma_S}{\vdash \forall x.P(x) \Rightarrow P(S(x))}}{\vdash \forall x.P(x)}\ (\textsc{Ind})$$

as an abreviation for the following situation:

$$\dfrac{\dfrac{\dfrac{\sigma_0}{\vdash P(0)} \qquad \dfrac{\sigma_S}{\vdash \forall x.P(x) \Rightarrow P(S(x))}}{\vdash\vdash P(0) \wedge \forall x.P(x) \Rightarrow P(S(x))} \qquad \dfrac{}{\vdash P(0) \wedge \forall x.P(x) \Rightarrow P(S(x)) \Rightarrow \forall x.P(x)}\ (\textsc{Ax})}{\vdash \forall x.P(x)}$$

Now consider:

$$\dfrac{\dfrac{\sigma_0}{\vdash P(0)} \qquad \dfrac{\sigma_S}{\vdash \forall x.P(x) \Rightarrow P(S(x))}}{\dfrac{\vdash \forall x.P(x)}{\vdash P(0)}\ (\forall\text{-E})}\ (\textsc{Ind})$$

This is the first possible form of an *induction cut* which can be obviously simplified into σ_0.

The second form is:

$$\dfrac{\dfrac{\sigma_0}{\vdash P(0)} \qquad \dfrac{\sigma_S}{\vdash \forall x.P(x) \Rightarrow P(S(x))}}{\dfrac{\vdash \forall x.P(x)}{\vdash P(S(t))}\ (\forall\text{-E})}\ (\textsc{Ind})$$

where t can be any term. This cut can be transformed into:

$$\dfrac{\dfrac{\sigma_S}{\vdash \forall x.P(x) \Rightarrow P(S(x))}}{\vdash P(t) \Rightarrow P(S(t))} \qquad \dfrac{\dfrac{\sigma_0}{\vdash P(0)} \qquad \dfrac{\sigma_S}{\vdash \forall x.P(x) \Rightarrow P(S(x))}}{\dfrac{\vdash \forall x.P(x)}{\vdash P(t)}\ (\forall\text{-E})}\ (\textsc{Ind})$$
$$\vdash P(S(t))$$

Note that this last cut-elimination is less a "simplification": the transformed proof is larger than the original one. However, we see that induction is now used for a smaller natural number: t in place of $S(t)$.

We will state and prove properties for proofs in arithmetic which are also free of axiomatic cuts. This will be particularly useful once we will have defined arithmetic as a theory modulo some rewrite rules, using the techniques of the next chapter.

Chapter 4

Deduction modulo

A first step in making the formalism more comfortable 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:

$$(\textsc{Conv})\frac{\Gamma \vdash A}{\Gamma \vdash B} \text{ if } A =_R B$$

In general we want the relation $=_R$ to be decidable in order to ba able to check the validity of a proof-tree.

More precisely, we will consider that $=_R$ is the reflexive, symmetric and transitive closure of a rewrite relation $\triangleright_R$ with the following properties:

1. $\triangleright_R$ is terminating and confluent (hence every proposition has a unique normal form),
2. A proposition $A \wedge B$ (resp. $A \vee B, A \Rightarrow B, \forall x.A, \exists x.A$) only rewrites to propositions $A' \wedge B'$ (resp. $A' \vee B', A' \Rightarrow B', \forall x.A', \exists x.A'$) with $A \triangleright_R A'$ or $B \triangleright_R B'$.
3. The proposition $\bot$ is normal.

This implies that the relation $=_R$ is well-behaved, in the following sense:

Condition 1. *We require that, (1) $=_R$ is the transitive, reflexive, symmetric closure of a confluent rewrite system and (2) that for any propositions A, B, C:*

- *If $C =_R A \wedge B$ (resp. $A \vee B$, $A \Rightarrow B$) then either:*
  - *C is atomic,*
  - *or $C = A' \wedge B'$ (resp. $A' \vee B'$, $A' \Rightarrow B'$) with $A =_R A'$ and $B =_R B'$.*

- *If* $C =_R \forall x.A$ *(resp.* $\exists x.A$*) then either:*
  - C *is atomic,*
  - *or* $C = \forall x.A'$ *(resp.* $\exists x.A'$*) with* $A =_R A'$*.*

We can check that the definition of arithmetic given below verifies these conditions.

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:

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

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:

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

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.

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:

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

  where $\top$ can stand for any easily provable proposition ($0 = 0$, $\bot \Rightarrow \bot$...)

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.

Note that we now have a presentation of arithmetic with just the reflexivity axiom and two axiom schemes:

$$\begin{gathered} \forall x. x = x \\ \forall x. \forall y. P(x) \wedge x = y \Rightarrow P(y) \\ P[x \mapsto 0] \wedge (\forall x. P \Rightarrow P[x \mapsto S(x)]) \Rightarrow \forall x. P. \end{gathered}$$

This will allow us to make some interesting observations regarding cut-free proofs.

4.3 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. $(S\ (S\ (S\ (S\ 0)))))$.

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. $(S\ (S\ (S\ (S\ 0)))))$.

4.4 Cuts in Deduction Modulo

One important nice property of deduction modulo is that it allows to make more cuts visible. The main point is that we rephrase natural deduction modulo, by dropping the explicit rule given at the beginning of the chapter

$$\frac{\Gamma \vdash A}{\Gamma \vdash B} \text{ if } A =_R B \ \ (\textsc{Conv})$$

and instead we add the possibility to perform rewriting steps inside any of the regular natural deduction rules. For instance, the introduction rule for conjunction becomes:

$$\text{if } C =_R A \wedge B \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash C} \ (\wedge\text{-I})$$

In other words, the rewrite steps do not appear anymore as node in the proof/derivation.

This will be more apparent when we will look at the axiomatic cuts in arithmetic.

Lemma 4.4.1 (Cut free proofs modulo)**.** *A proof of* $[] \vdash A$ *in deduction modulo always ends with an introduction rule.*

Proof. The condition on $=_R$ allows to perform the same induction over the derivation than for regular natural deduction.

- The proof cannot end with the axiom rule because there are no hypotheses.
- If the proofs and with $\wedge$-E_1

$$\dfrac{\dfrac{\sigma}{\vdash A \wedge B}}{\vdash A}$$

  then, by induction hypothesis, σ ends with an introduction rule. Because of condition 1, this rule must be $\wedge$-I. So the proof is not cut-free.

The other connectives are treated similarly. □

4.4.1 Cut free proofs in Heyting Arithmetic

We can now consider cut-free proofs in the presentation of arithmetic given in 4.2. The nice point is that we have a notion of axiomatic cut for every remaining axiom:

- The equality cut deals with the reflexivity axiom and the Leibniz scheme. The first plays the role of an introduction rule, the second plays the role of the elimination rule.
- The induction cut allows to eliminate induction from proofs when they are applied to closed natural numbers.

Theorem 4.4.2. *Consider a closed proposition A in the language of arithmetic (without free variables). Any cut free proof of A in Heyting arithmetic verifies one of the following properties:*

1. *It ends with an introduction rule.*
2. *It ends with an axiom rule.*
3. *It is a proof of $n = n$ obtained by specializing the reflexivity axiom for some closed numeral n (that is $S^{(n)}(0)$).*

4. *It is a partial application of Leibniz scheme where one or the two universal quantifiers have been eliminated. That is A is of the form $\forall y.n = y \wedge P(n) \Rightarrow P(y)$ or $n = y \wedge P(n) \Rightarrow P(m)$.*

5. *It is a partial application of the induction scheme, but where the last universal quantification has not been instantiated. That is A is of one of the following forms:*

   - $(\forall x.P(x) \Rightarrow P(S(x))) \Rightarrow \forall x.P(x)$
   - $\forall x.P(x)$

Proof. We can start by noticing that any closed object t rewrites to a numeral $S^{(n}(0)$. This is an elementary combinatorial result from the fact that the rewrite rules are confluent and terminating.

From this, we can deduce that a proof of $t = u$ verifying the conditions stated in the theorem is necessarily obtained using the reflexivity axiom (and thus t and u are identical modulo rewriting).

The theorem itself is then proved by induction over the structure (or equivalently the size) of the cut free proof of A.

If the cut free proof ends with an introduction or an axiom rule, then the property holds.

Suppose the cut free proofs ends with an elimination rule. Then its main premise cannot end with an introduction rule; it thus ends with either an axiom or another elimination rule.

If it ends with an axiom, we are in one of the foreseen cases. If it ends with an introduction rule, we can apply the induction hypothesis to the premise of the elimination rule. By iterating this, we must end up finding an axiom rule. The axiom being either reflexivity, Leibniz or induction.

We are thus in one of the following cases:

Reflexivity The whole proof if of the form

$$\dfrac{\dfrac{}{\vdash_{HA} \forall x.x = x}(\text{Ax})}{\vdash_{HA} t = t}\ (\forall\text{-E})$$

which is one of the foreseen cases.

Leibniz If there are one or two elimination rules after the axiom rule, we are in case 4 of the theorem. If there a three or more elimination rules, the proof or a subtree of the proof is of the following form:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash_{HA} \forall x.\forall y.(x = y \wedge P(x)) \Rightarrow P(y)}(\text{Ax})}{\vdash_{HA} \forall y.(t = y \wedge P(t)) \Rightarrow P(y)}}{\vdash_{HA} t = u \wedge P(u) \Rightarrow P(t)} \qquad \dfrac{\sigma}{\vdash_{HA} t = u \wedge P(u)}}{\vdash_{HA} P(t)}$$

But then the induction hypothesis ensures that σ ends with an introduction rule. It thus contains a closed cut free proof of $t = u$, which must itself be obtained by reflexivity. We thus have an equality cut, so this situation cannot happen.

Induction The situation is similar to the previous case. We let you write the details.

□

Chapter 5

Cut Elimination

We have seen that there is a terminating strategy to eliminate cuts in proofs of first order logic: by always starting by an innermost cut of largest weight. Actually, any strategy will terminate. That is if one takes a proof, reduces any cut and iterates, there will be no cut left after some time.

5.1 Logical Cuts in Natural Deduction

It is *not* obvious that the process of applying these transformation terminates ending with a cut-free proof. One main point is that replacing an axiom rule in a derivation (lemma 3.7.2) can create a new cut when the derivation replacing the axiom rule ends with an introduction rule.

We show this termination by reusing strong normalization (Theorem 2.5.7) of the simply typed λ-calculus (with sums and products) of chapter 2. This is a first use of the Curry-Howard correspondence - which will be detailed further in coming chapters. We interpret every proposition by a type, every derivation by a term, and we check that a cut-elimination step is mimicked by a $\triangleright$-reduction step.

Types. Given a proposition A, we define a type $|A|$ by induction on the structure of A:

- $|A| \equiv \mathsf{unit}$ if A is atomic (this includes false),
- $|A \wedge B| \equiv |A| \times |B|$,
- $|A \vee B| \equiv |A| + |B|$,
- $|A \Rightarrow B| \equiv |A| \to |B|$,
- $|\forall x.\, A| \equiv N \to |A|$,
- $|\exists x.\, A| \equiv N \times |A|$.

Note that $|A|$ does not depend on x: this is a genuine simplification with respect to the dependently typed reading treated later. It will however be sufficient for ensuring cut elimination.

Terms. To a context $\Gamma = A_1, \ldots, A_n$ we associate the a sequence of typed variables: $x_1^{|A_1|}, \ldots, x^{|A_n|}$ (one fresh variable per hypothesis). We then define $\|d\|$, a term of type $|A|$, by induction on a derivation d of $\Gamma \vdash A$:

- $\|\mathrm{Ax}(A)\| \equiv x_i^A$, the variable associated to $A \in \Gamma$;
- $\|\wedge\text{-I}(d_1, d_2)\| \equiv (\|d_1\|, \|d_2\|)$, and $\|\wedge\text{-E}_1(d)\| \equiv \pi_1(\|d\|)$, $\|\wedge\text{-E}_2(d)\| \equiv \pi_2(\|d\|)$;
- $\|\vee\text{-I}_1(d)\| \equiv i(\|d\|)$, $\|\vee\text{-I}_2(d)\| \equiv j(\|d\|)$, and $\|\vee\text{-E}(d, d_1, d_2)\| \equiv \delta(\|d\|, x^{|A|}.\|d_1\|, x^{|B|}.\|d_2\|)$ (using the δ eliminator of sum types);
- $\|\Rightarrow\text{-I}(d)\| \equiv \lambda x^{|A|}.\|d\|$, and $\|\Rightarrow\text{-E}(d_1, d_2)\| \equiv (\|d_1\|\ \|d_2\|)$;
- $\|\forall\text{-I}(d)\| \equiv \lambda x^N.\|d\|$ (legitimate precisely because the rule requires $x \notin \mathrm{FV}(\Gamma)$), and $\|\forall\text{-E}(d, t)\| \equiv (\|d\|\ \ulcorner t \urcorner)$;
- $\|\exists\text{-I}(d, t)\| \equiv (\ulcorner t \urcorner, \|d\|)$, and $\|\exists\text{-E}(d, d')\| \equiv \|d'\|[x \backslash \pi_1(\|d\|)][y \backslash \pi_2(\|d\|)]$, where x is the (individual) variable and y the hypothesis variable bound by the rule;
- $\|\mathsf{false}\text{-E}_B(d)\| \equiv c_B$, a fixed variable of type $|B|$ chosen once and for all for each formula B — *ignoring* $\|d\|$ entirely.

Remark. For an arbitrary theory, the term $\ulcorner t \urcorner : N$ associated to a first-order term t can simply be taken to be 0 for every t: since $|A|$ never depends on the individuals it quantifies over, no case below actually needs $\ulcorner t \urcorner$ to carry any real information — only termination is at stake here. A faithful (computationally meaningful) translation $\ulcorner \cdot \urcorner$ is given in **??** once we fix the theory of arithmetic; it becomes useful there to read off actual witnesses from cut-free proofs (constructivity, see **??**).

Theorem 5.1.1 (Simulation)**.** *If d reduces to d' by one of the cut-elimination steps of Figure 3.3 then $\|d\| \rhd \|d'\|$.*

Proof. By cases on the connector of the cut, as in Figure 3.3.

- $\Rightarrow$ Already computed above: $\|d\| = (\lambda x^{|A|}.\|\sigma\|)\ \|\tau\| \rhd \|\sigma\|[x \backslash \|\tau\|] = \|\sigma[A \backslash \tau]\| = \|d'\|$.
- $\wedge$ $\|d\| = \pi_i(\|d_1\|, \|d_2\|) \rhd \|d_i\| = \|d'\|$, the projection rule of product types.
- $\vee$ $\|d\| = \delta(i(\|d_1\|), x.\|e_1\|, x.\|e_2\|) \rhd \|e_1\|[x \backslash \|d_1\|] = \|d'\|$ (and symmetrically for the second injection), the reduction rule of the δ eliminator.
- $\forall$ $\|d\| = (\lambda x^N.\|d_1\|)\ \ulcorner t \urcorner \rhd \|d_1\|[x \backslash \ulcorner t \urcorner] = \|d_1[x \backslash t]\| = \|d'\|$.
- $\exists$ $\|d\| = \|d_2\|[x \backslash \pi_1(\ulcorner t \urcorner, \|d_1\|)][y \backslash \pi_2(\ulcorner t \urcorner, \|d_1\|)] \rhd \|d_2\|[x \backslash \ulcorner t \urcorner][y \backslash \|d_1\|] = \|d'\|$, the two projections of the pairing $(\ulcorner t \urcorner, \|d_1\|)$ reducing on their constructor.

We can remark that the false-E rule cannot create a cut, so it is not a problem that its interpretation c_B never contributes to a redex. □

Corollary 5.1.2 (Termination of logical cut-elimination)**.** *Every process of cut-elimination on a natural deduction proof terminates, i.e. every proof of* $\Gamma \vdash A$ *reduces (in finitely many steps) to a cut-free proof of* $\Gamma \vdash A$*.*

Proof. An infinite cut-elimination sequence $d_0 \rightsquigarrow d_1 \rightsquigarrow \cdots$ would give, by Theorem 5.1.1, an infinite reduction sequence $\|d_0\| \rhd \|d_1\| \rhd \cdots$ (dropping the false-E steps, which by the remark above cannot occur infinitely often in a row without an intervening genuine cut), contradicting the strong normalization propertyTheorem 2.5.7. □

5.2 Axiomatical Cuts in Arithmetic

We will now extend our interpretation of proofs by typed λ-terms by translations proofs of Heyting Arithmetic in FOL modulo into terms typable in System T. We keep the translation of proposition to types of the previous section. We need to interpret the remaining axioms, that is reflexivity and the Leibniz and induction schemes.

- An axiomatic proof of $\forall x.x = x$ is interpreted by $\lambda x^N.()$
- An axiomatic proof of $\forall xy.x = y \Rightarrow P(x) \Rightarrow P(y)$ is interpreted by $\lambda x^N.\lambda y^N.\lambda e^{\mathsf{unit}}.\lambda p^{|P(x)|}.\mathsf{check}\ e.p.$
- An axiomatic proof of

  $$P(0) \Rightarrow (\forall y.P(y) \Rightarrow P(S(y))) \Rightarrow \forall x.P(x)$$

  is interpreted by:

  $$\lambda p_0^{|P|}.\lambda p_S^{N\to|P|\to|P|}.\lambda x^n.R_{|P|}(p_0, p_S, x).$$

  We leave it to the reader to verify that with this extended interpretation, if a proof derivation σ is transformed into σ' by the elimination of an equality of induction cut, then $||\sigma|| \rhd_\beta^+ ||\sigma'||$. That is we may need more than one β-reduction to mimick the cut elimination step. But strong normalization of the λ-calculus still ensures the termination of cut elimination for Heyting Arithmetic.

  Corollary 5.2.1. *A proof* $\vdash A$ *of any proposition* A *of arithmetic, using only the three axiom schemes (reflexity, Leibniz and induction) can be transformed into a proof free of logical and axiomatic cuts.*

  Corollary 5.2.2. *A proof* $\vdash A \vee B$ *for any propositions* A *and* B *of arithmetic, using only the three axiom schemes, can be transformed into a cut-free proof ending with an introduction rule. Thus it contains either a proof of* A *or a proof of* B*.*

Corollary 5.2.3. *A proof $\vdash \exists x.A$ for any proposition A of arithmetic, using only the three axiom schemes, can be transformed into a cut-free proof ending with an introduction rule. Thus it contains an object t of arithmetic and a proof of $A[x \setminus t]$.*

5.3 A Realizability Interlude

[BW : I want to go over this part which is not considered finished]

Associating computational content to proofs is not tied to typing at all. The oldest such construction, Kleene's *realizability* (1945), predates any λ-calculus-based reading of proofs as terms — it is in fact older than the Curry-Howard correspondence itself, and its original version does not use λ-terms but indices of partial recursive functions. What follows is a λ-calculus rendering of the same idea, reusing the untyped (type-erased) version of the term language of the previous chapters.

Unlike Theorem 5.1.1, this gives a route to consistency and to extracting constructive content that never goes through cut-elimination or a normal form: it applies directly to an arbitrary derivation. It is also, on purpose, a coarser tool than typing: as the remark on Markov's principle below illustrates, it validates some principles that are not derivable at all. Realizability is sound with respect to provability, but — unlike the Heyting-algebra semantics dual to it, which is complete and is the standard tool to show formulas are *not* derivable — it is not complete: it may validate strictly more.

5.3.1 Untyped terms and the realizability relation

We reuse the raw term grammar of the previous chapters (λ-abstraction and application, pairing (t, u) and its projections, injections $i(t), j(t)$ and their eliminator δ, the unit element $()$, 0, S, and the recursor $R(t_0, t_S, t)$), simply forgetting all typing information: from now on a realizer can be any raw term, whether or not it is typable in any of the systems considered so far. As before, $\mathcal{I}$ denotes a valuation of individual variables by natural numbers, and $\mathcal{I}[x \mapsto n]$ the valuation agreeing with $\mathcal{I}$ except that it sends x to n.

Definition 5.3.1 (Realizability)**.** For every proposition A and valuation $\mathcal{I}$, we define a relation $t \Vdash_{\mathcal{I}} A$ between (untyped) terms t and A, by induction on the structure of A:

- no term realizes false: $t \Vdash_{\mathcal{I}} \mathsf{false}$ never holds;
- $t \Vdash_{\mathcal{I}} A \wedge B$ iff $t \rhd^* (u, v)$ for some $u \Vdash_{\mathcal{I}} A$ and $v \Vdash_{\mathcal{I}} B$;
- $t \Vdash_{\mathcal{I}} A \vee B$ iff either $t \rhd^* i(u)$ for some $u \Vdash_{\mathcal{I}} A$, or $t \rhd^* j(v)$ for some $v \Vdash_{\mathcal{I}} B$;
- $t \Vdash_{\mathcal{I}} A \Rightarrow B$ iff for every $u \Vdash_{\mathcal{I}} A$, $(t\ u) \Vdash_{\mathcal{I}} B$;
- $t \Vdash_{\mathcal{I}} \forall x.\, A$ iff for every $n \in \mathbb{N}$, $(t\ S^{(n)}(0)) \Vdash_{\mathcal{I}[x \mapsto n]} A$;
- $t \Vdash_{\mathcal{I}} \exists x.\, A$ iff $t \rhd^* (S^{(n)}(0), u)$ for some $n \in \mathbb{N}$ and some $u \Vdash_{\mathcal{I}[x \mapsto n]} A$;

- for an atomic proposition $P(a_1, \ldots, a_k)$ (equality included), $t \Vdash_{\mathcal{I}} P(a_1, \ldots, a_k)$ iff $t \rhd^* ()$ and $P(a_1[\mathcal{I}], \ldots, a_k[\mathcal{I}])$ genuinely holds, as a fact about natural numbers in the standard model — e.g. $t \Vdash_{\mathcal{I}} a = b$ iff $t \rhd^* ()$ and $a[\mathcal{I}] = b[\mathcal{I}]$, and likewise for any base predicate introduced by rewriting, such as D of § 4.2.

Remark. Compare with the reducibility candidates $|U \times V|, |U + V|$ of the previous chapter (Definition 2.3.1, extended to products and sums): the clauses above are exactly the same "t reduces to a term of the right shape" conditions, *minus* the requirement $t \in \mathsf{SN}$. Dropping strong normalization is precisely what lets realizers escape the typed calculi considered so far, and it is also what will let us realize a principle such as Markov's below, whose only witnesses are not strongly normalizing terms.

Lemma 5.3.1 (Closure under reduction)**.** *For every A, $\mathcal{I}$, and $t \rhd t'$:*

1. *if $t \Vdash_{\mathcal{I}} A$ then $t' \Vdash_{\mathcal{I}} A$;*
2. *if $t' \Vdash_{\mathcal{I}} A$ then $t \Vdash_{\mathcal{I}} A$.*

Proof. By induction on A. For $\wedge, \vee, \exists$ and the atomic case, both directions are immediate from transitivity of $\rhd^*$: a witness reduct of t' (resp. of t) is still reachable from t (resp. gives one from t' once the first step is dropped, using that $t \rhd t'$ contributes nothing to reaching an *already reached* reduct of t'). For $\Rightarrow$ and $\forall$: reduction is a congruence for application, so $t \rhd t'$ gives $(t\ u) \rhd (t'\ u)$ for every relevant u; conclude by the induction hypothesis on B (resp. on A) at this reduction step. Note, in contrast with Theorems 2.5.5 and 2.6.1, that direction 2 needs no condition on t being neutral: realizability only ever asks for the *existence* of a reduct of the right shape, and prefixing by one more reduction step never removes such a reduct. □

5.3.2 Adequacy

We reuse, unchanged, the term assignment $\|d\|$ of § 5.1, now simply read as an untyped term (the type superscripts on bound variables carry no meaning below and could be erased).

Theorem 5.3.2 (Adequacy)**.** *Let d be a derivation of $A_1, \ldots, A_n \vdash A$, with associated hypothesis variables $x_1, \ldots, x_n$. For every valuation $\mathcal{I}$ and every $t_1, \ldots, t_n$ with $t_i \Vdash_{\mathcal{I}} A_i$ for all i:*

$$\|d\|[x_1 \backslash t_1, \ldots, x_n \backslash t_n] \Vdash_{\mathcal{I}} A.$$

Proof. By induction on d. Write σ for the substitution $[x_1 \backslash t_1, \ldots, x_n \backslash t_n]$ throughout. Most cases are exactly the computation already carried out in the proof of Theorem 5.1.1, now read as producing a realizer instead of preserving a type:

Ax $\|\mathrm{Ax}(A)\|\sigma = t_i \Vdash_{\mathcal{I}} A_i = A$ directly, by hypothesis.

$\wedge$ $\|\wedge\text{-I}(d_1, d_2)\|\sigma = (\|d_1\|\sigma, \|d_2\|\sigma)$ realizes $A \wedge B$ at once (zero reduction steps needed), using the induction hypothesis on d_1, d_2. Conversely, if $\|d\|\sigma \Vdash_{\mathcal{I}} A \wedge B$, i.e. $\|d\|\sigma \rhd^* (u, v)$ with $u \Vdash_{\mathcal{I}} A$, $v \Vdash_{\mathcal{I}} B$, then $\pi_1(\|d\|\sigma) \rhd^* u$ (prefixing by the

same reduction, then the projection rule), so $\|\wedge\text{-E}_1(d)\|\sigma = \pi_1(\|d\|\sigma) \Vdash_{\mathcal{I}} A$, and symmetrically for π_2.

$\vee$ Symmetric to $\wedge$, using the injections and the reduction rule of δ.

$\Rightarrow$ $\|\Rightarrow\text{-I}(d)\|\sigma = \lambda x^{|A|}.(\|d\|\sigma)$: for any $u \Vdash_{\mathcal{I}} A$, $(\lambda x.\|d\|\sigma)\ u \rhd \|d\|\sigma[x\backslash u]$, which realizes B by the induction hypothesis on d (extending σ by $x\backslash u$, legitimate since $u \Vdash_{\mathcal{I}} A$); conclude by Theorem 5.3.1. For $\|\Rightarrow\text{-E}(d_1, d_2)\|\sigma = (\|d_1\|\sigma\ \|d_2\|\sigma)$: by the induction hypothesis $\|d_1\|\sigma \Vdash_{\mathcal{I}} A \Rightarrow B$ and $\|d_2\|\sigma \Vdash_{\mathcal{I}} A$, so directly $(\|d_1\|\sigma\ \|d_2\|\sigma) \Vdash_{\mathcal{I}} B$ by definition of the $\Rightarrow$ clause.

$\forall$ $\|\forall\text{-I}(d)\|\sigma = \lambda x^N.(\|d\|\sigma)$ (legitimate since $x \notin \mathrm{FV}(\Gamma)$, so x does not occur in σ): for every n, $(\lambda x.\|d\|\sigma)\ S^{(n)}(0) \rhd \|d\|\sigma$, which realizes A at $\mathcal{I}[x \mapsto n]$ by the induction hypothesis on d applied at this extended valuation — this is exactly where the individual variable x acquires its meaning. Conclude by Theorem 5.3.1. For $\|\forall\text{-E}(d, t)\|\sigma = (\|d\|\sigma\ \ulcorner t \urcorner)$: writing $n = t[\mathcal{I}]$, faithfulness of $\ulcorner . \urcorner$ gives $\ulcorner t \urcorner \rhd^* S^{(n)}(0)$, so by the induction hypothesis (which gives, at $\mathcal{I}[x \mapsto n]$, a realizer of A, i.e. of $A[x\backslash t]$ at $\mathcal{I}$) and Theorem 5.3.1 again, $(\|d\|\sigma\ \ulcorner t \urcorner) \Vdash_{\mathcal{I}} A[x\backslash t]$.

$\exists$ Symmetric to $\forall$, combining the $\wedge$ and $\exists$ clauses as $\|\exists\text{-I}(d, t)\|$ and $\|\exists\text{-E}(d, d')\|$ already do in § 5.1.

Two cases are genuinely new with respect to Theorem 5.1.1, and deserve a full argument rather than a reference to it.

false-E. Here $\|\mathsf{false}\text{-E}_B(d)\|\sigma = c_B$, the fixed free variable of § 5.1, entirely independent of d and of σ. Unlike in the typed argument, where c_B trivially inhabited $|B|$ by property 3 of Theorem 2.5.5 applied to a variable (which has no reduct at all, so the universal condition of that clause holds vacuously), a bare variable realizes *nothing* here: realizability only ever asks for the *existence* of a reduct of the right shape, and a variable has none. The case is handled differently: d itself is a (strictly smaller) derivation of $\Gamma \vdash \mathsf{false}$, so the induction hypothesis applied to d says that $t_1, \ldots, t_n$ realizing Γ would give $\|d\|\sigma \Vdash_{\mathcal{I}} \mathsf{false}$ — which is impossible, as no term realizes false. So the hypothesis "$t_1, \ldots, t_n$ realize Γ" is itself contradictory whenever $\Gamma \vdash \mathsf{false}$ is derivable, and the statement to prove for this case, namely that this same hypothesis entails $\|\mathsf{false}\text{-E}_B(d)\|\sigma \Vdash_{\mathcal{I}} B$, holds vacuously: it is never actually called upon.

Ind. Recall the induction axiom (Induction Cuts, previous chapter), abbreviating the derivation of $\vdash \forall x.\, P(x)$ from $\sigma_0 \vdash P(0)$ and $\sigma_S \vdash \forall x.\, P(x) \Rightarrow P(S(x))$. Set

$$\|\mathrm{Ind}(P, \sigma_0, \sigma_S)\| \equiv \lambda x^N.\, R(\|\sigma_0\|, \|\sigma_S\|, x).$$

Write $t_0 = \|\sigma_0\|\sigma$, $t_S = \|\sigma_S\|\sigma$. We must show, for every n, that $(\lambda x.R(t_0, t_S, x))\ S^{(n)}(0)$ realizes P at $\mathcal{I}[x \mapsto n]$; by Theorem 5.3.1 it is enough to show $R(t_0, t_S, S^{(n)}(0)) \Vdash_{\mathcal{I}[x \mapsto n]} P$, which we do by (meta-level) induction on n — nested *inside* the induction on d, and genuinely using that $\mathbb{N}$ is well-founded, not just that the derivation is a finite tree:

- $n = 0$: $R(t_0, t_S, 0) \rhd t_0$, and $t_0 \Vdash_{\mathcal{I}[x\mapsto 0]} P$ by the (outer) induction hypothesis applied to the strictly smaller derivation σ_0. Conclude by Theorem 5.3.1.
- $n = k + 1$: $R(t_0, t_S, S^{(k+1)}(0)) \rhd (t_S\ S^{(k)}(0)\ R(t_0, t_S, S^{(k)}(0)))$. By the outer induction hypothesis applied to σ_S, $t_S \Vdash_{\mathcal{I}} \forall x.\, P(x) \Rightarrow P(S(x))$, i.e. for every m, $(t_S\ S^{(m)}(0)) \Vdash_{\mathcal{I}[x\mapsto m]} P \Rightarrow P(S(x))$. Take $m = k$ and, as the realizer of the antecedent P at $\mathcal{I}[x \mapsto k]$, the term $R(t_0, t_S, S^{(k)}(0))$, which realizes it by the (meta-)induction hypothesis on k. This gives $(t_S\ S^{(k)}(0)\ R(t_0, t_S, S^{(k)}(0))) \Vdash_{\mathcal{I}[x\mapsto k+1]} P$, and conclude again by Theorem 5.3.1.

This completes the induction on d. □

5.3.3 Corollaries

Corollary 5.3.3 (Consistency)**.** *There is no derivation of* $\vdash \mathsf{false}$ *(from no hypotheses, in particular in Heyting Arithmetic).*

Proof. If d derived $\vdash \mathsf{false}$, Theorem 5.3.2 (with $n = 0$, so its hypothesis is void) would give $\|d\| \Vdash_{\mathcal{I}} \mathsf{false}$ for any $\mathcal{I}$ — impossible, as no term realizes false. □

Corollary 5.3.4 (Constructive content)**.** *If d derives $\vdash A \vee B$ (no hypotheses, both closed — so the choice of $\mathcal{I}$ below is immaterial), then $\|d\|$ reduces to $i(u)$ with $u \Vdash_{\mathcal{I}} A$, or to $j(v)$ with $v \Vdash_{\mathcal{I}} B$ — one can* compute*, by reducing $\|d\|$, which disjunct holds and a realizer of it. Likewise, if d derives $\vdash \exists x.\, A$, then $\|d\|$ reduces to $(S^{(n)}(0), u)$ for an actual $n \in \mathbb{N}$ with $u \Vdash_{\mathcal{I}[x\mapsto n]} A$: an explicit witness, computed rather than merely known to exist.*

Proof. Immediate instance of Theorem 5.3.2 ($n = 0$ hypotheses) unfolded at the definition of $\Vdash$ for $\vee$, resp. $\exists$. □

Remark. This is the same disjunction and existence property already obtained, by a different (syntactic, subformula-property) route, in § 5.1 of the previous chapter — but this time without ever eliminating a single cut, or invoking Theorems 2.5.7 and 2.6.3 at all. It is a genuinely independent argument, not merely an alternative presentation of the same one.

Remark (Markov's principle)*.* This independence from typing lets realizability validate principles that are *not* derivable in Heyting Arithmetic — illustrating that realizability, unlike the (complete) Heyting-algebra semantics dual to it, is sound but not complete. A standard example is *Markov's principle*: for A decidable (say, primitive recursive),

$$\big(\forall x.\, A(x) \vee \neg A(x)\big) \wedge \neg\neg\exists x.\, A(x) \Rightarrow \exists x.\, A(x).$$

Given a decision term for A, build the unbounded search

$$\mathit{search} := \mu n.\, A(n)$$

(recursively test $A(0), A(1), \ldots$ and stop at the first success): this is *not* a term of System T, or of any strongly normalizing calculus — it is a genuine fixed point, and this is exactly why Markov's principle is not provable in HA (derivability goes through Theorem 5.1.1, hence through SN terms only, by Theorem 2.6.3). Yet *search does* realize Markov's principle: given a realizer of the double negation $\neg\neg\exists x.\, A(x)$, one shows *search* terminates by a classical (meta-level) argument — if it did not, $\forall x.\, \neg A(x)$ would genuinely hold, from which one builds a realizer of $\neg\exists x.\, A(x)$, which composed with the given realizer of $\neg\neg\exists x.\, A(x)$ would realize false, impossible by Theorem 5.3.3. So *search* terminates, and its result is exactly the required witness. Note that this termination argument is itself classical (by contradiction) — entirely legitimate, since it is ordinary mathematics about the realizability relation, not a derivation internal to HA.

Chapter 6

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.

6.1 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.

6.2 The objects of HOL

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

Definition 6.2.1 (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 6.2.2. The raw terms are built defined by the following grammar:

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

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:

$$\begin{array}{lcllcl} 0 & : & \iota & + & : & \iota \to \iota \to \iota \\ S & : & \iota \to \iota & \times & : & \iota \to \iota \to \iota \end{array}$$

and for o:

$$\begin{array}{lcl} \Rightarrow & : & o \to o \to o \\ \forall_T & : & (T \to o) \to o \end{array}$$

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 $(\forall_T\ \lambda x^T.P)$.

6.3 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.

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

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

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

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

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

6.4 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 $\Rightarrow$. The key is the very powerful ability to quantify over all propositions ($\forall_o$) to form a new proposition.

6.4.1 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 $\bot \Rightarrow P$.

6.4.2 Conjuction

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

6.4.3 Disjunction

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

6.4.4 Existential quantifier

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

6.5 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.

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

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:

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

6.6 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.

6.7 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:

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

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 :

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

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 $x_1, x_2, \ldots$ such that

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

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.
```

6.8 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:

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

So (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:

$$\forall x^{\iota}.(\text{Nat}\ n^{\iota}) \Rightarrow n^{\iota} + 0 = n^{\iota}$$

6.9 Normal terms in HOL

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

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

6.10 Functions in HOL

This shows that 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.

6.10.1 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, $P(x) \Rightarrow P(\varepsilon(P))$.

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

6.10.2 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 principle of Hilbert's operator:

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

6.10.3 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:

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

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:

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

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

6.10.4 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 $A \Rightarrow B$.

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 7

Dependent Types

7.1 Definition

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

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

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

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

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

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

We will see that:

- If $\mathcal{R} = \{(\mathsf{Prop}, \mathsf{Prop})\}$ then it is the simply typed calculus.
- 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 add $(\mathsf{Kind}, \mathsf{Prop})$ we get impredicativity. $\mathcal{R} = \{(\mathsf{Prop}, \mathsf{Prop}); (\mathsf{Kind}, \mathsf{Prop})\}$ gives the system F.
- 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 defined the calculus of constructions (CoC) which is the core of Coq.

7.2 Basic properties

Although some properties vary greatly with $\mathcal{R}$, the basic metatheory is the same. These lemmas are all done by simple inductions over the typing judgements, but it is important to do them in the right order.

Lemma 7.2.1 (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 7.2.2 (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 7.2.3. *1. If* $\Gamma \vdash t : T$ *is derivable, then* Γ wf *is derivable.*

2. If $\Gamma \vdash t : T$ *is derivable, then either* $T = \mathsf{Kind}$ *or* $\Gamma \vdash T : s$ *is derivable for some sort* s*.*

Lemma 7.2.4. *If* $\Gamma \vdash \forall x : A.B : s$ *then* $\Gamma(x : A) \vdash B : s$*.*

The following lemma is technically important. It basically states that the derivation of $\Gamma \vdash t : T$ is of the same shape as t.

Lemma 7.2.5 (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 $\Gamma \vdash forallx : A.B : T$ *then* T *is either* Prop *or* Kind *and* $\Gamma(x : A) \vdash B : T$*.*

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

Corollary 7.2.6 (Type uniqueness)**.** *If* $\Gamma \vdash t : A$ *and* $\Gamma \vdash t : B$ *are derivable, then* $A =_\beta B$*.*

Lemma 7.2.7 (Subject reduction)**.** *If* $\Gamma \vdash t : A$ *and* $t \rhd_\beta t'$ *(resp.* $A \rhd_\beta A'$*) then* $\Gamma \vdash t' : A$ *(resp.* $\Gamma \vdash t : A'$*).*

7.3 Normalization

Fundamentally, the normalization property for LF is not difficult. We will map the system to simply typed λ-calculus by erasing type dependency.

Lemma 7.3.1 (Stratification)**.** *If* $\Gamma \vdash t : T$*, then we have one of the following situations:*

- $T = \mathsf{Kind}$*,*
- *or* $\Gamma \vdash T : \mathsf{Kind}$ *in which case we say that* T *is a kind and* t *is a predicate,*

- *or* $\Gamma \vdash T : \mathsf{Prop}$ *in which case we say that* t *is a proof.*

Note that, to be precise, being a kind, predicate or proof is with respect to the context Γ*.*

Lemma 7.3.2. *In a context* Γ*, kinds, predicates and proofs belong respectively to the following grammars:*

$$\begin{array}{rcl} K & ::= & \mathsf{Type} \mid \Pi x : T.K \\ T & ::= & X \mid \Pi x : T.T \mid \lambda x : T.T \mid (T\ t) \\ t & ::= & x \mid \lambda x : T.t \mid (t\ t) \end{array}$$

where X *stands for a variable whose type in* Γ *is a kind, and* x *for a variable whose type in* Γ *is a predicate.*

Proof. By induction over the typing judgement. □

7.3.1 Mapping predicates to simple types

$$\begin{array}{rcl} \overline{X} & \equiv & X \\ \overline{\Pi x : T.U} & \equiv & \overline{T} \to \overline{U} \\ \overline{\lambda x : T.U} & \equiv & \overline{U} \\ \overline{(T\ t)} & \equiv & \overline{T} \end{array}$$

$$\begin{array}{rcl} \overline{\mathsf{Type}} & \equiv & \alpha \\ \overline{\Pi x : T.K} & \equiv & \overline{T} \to \overline{K} \end{array}$$

$$\begin{array}{rcl} \overline{[]} & \equiv & [(\alpha : \mathsf{Type})] \\ \overline{\Gamma; (X : K)} & \equiv & \overline{\Gamma}; (X : \mathsf{Type}); (\underline{X} : \overline{K}) \\ \overline{\Gamma; (x : T)} & \equiv & \overline{\Gamma}; (x : T) \end{array}$$

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

$$\begin{array}{rcl}
\overline{\overline{X}} & \equiv & \underline{\underline{X}} \\
\overline{\overline{(T\ t)}} & \equiv & (\overline{\overline{T}}\ \overline{\overline{t}}) \\
\overline{\overline{\lambda x : T.U}} & \equiv & \lambda x : \overline{T}.(\lambda_ : \alpha.\overline{\overline{U}}\ \overline{\overline{T}})
\end{array}$$

Lemma 7.3.3. *If* $\Gamma \vdash T : \mathsf{Type}$ *and* $\Gamma \vdash U : \mathsf{Type}$ *and* $T =_\beta U$ *then* $\overline{T} = \overline{U}$.

Lemma 7.3.4. *If* $\Gamma \vdash t : T$ *(resp.* $\Gamma \vdash t : K$*) then* $\overline{\Gamma} \vdash \overline{\overline{t}} : \overline{T}$ *(resp.* $\overline{\Gamma} \vdash \overline{\overline{T}} : \overline{K}$*).*

Lemma 7.3.5. *If* $t \rhd t'$ *then* $\overline{\overline{t}} \rhd^+ \overline{\overline{t'}}$. *If* $T \rhd T'\ \overline{\overline{T}}$ *then* $\rhd^+ \overline{\overline{T'}}$.

Proof. One first notices that $\overline{\overline{t[x \setminus u]}} = \overline{\overline{t}}[x \setminus \overline{\overline{u}}]$.

The proof is then quite straightforward by induction over the structure of t. The important case is:

$$(\lambda x : T.u\ v) \rhd u[x \setminus v]$$

which corresponds to:

$$\begin{array}{rcl}
\overline{\overline{(\lambda x : T.u\ v)}} & = & (\lambda x : \overline{T}.(\lambda_ : \alpha.\overline{\overline{u}}\ \overline{\overline{T}})\ \overline{\overline{v}}) \\
& \rhd & (\lambda x : \overline{T}.\overline{\overline{u}}\ \overline{\overline{v}}) \\
& \rhd & \overline{\overline{u}}[x \setminus \overline{\overline{v}}] \\
& = & \overline{\overline{u[x \setminus v]}}.
\end{array}$$

□

Lemma 7.3.6. *If* $\Gamma \vdash t : T$ *(resp.* $\Gamma \vdash T : K$*) then* $\overline{\overline{t}}$ *(resp.* $\overline{\overline{T}}$*) is well-typed in simply typed* λ*-calculus of type* $\overline{T}$ *(resp.* $\overline{K}$*).*

Thus $\overline{\overline{t}}$ *(resp.* $\overline{\overline{T}}$*) is strongly normalizing.*

Theorem 7.3.7. *If* $\Gamma \vdash t : T$ *(resp.* $\Gamma \vdash T : K$*) then* t *(resp.* T*) is strongly normalizing.*

7.4 Type checking

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

Theorem 7.4.1. *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.

□

Chapter 8

Martin-Löf's Type Theory

8.1 Introduction

This chapter gives one possible presentation of Martin-Löf's type theory. In substance, this formalism merges the various features we have studied so far:

- Dependent types and thus the Curry-Howard isomorphism,
- full first-order logic,
- arithmetic,
- a terminating but relatively expressive typed functional programming language,
- constructivity by combining intuitionistic logic and termination/cut elimination.

8.2 The syntax

Type theory terms are obtained by extending the terms for dependent types with:

- Natural numbers and recursor,
- sum types,
- dependent product types, also called Σ-types,
- the equality relation.

This gives the following grammar:

$$\begin{aligned} t \quad ::= \quad & x \mid \mathsf{Type} \mid \mathsf{Kind} \mid \lambda x : t.t \mid (t\ t) \mid \Pi x : t.t \mid \Sigma x : t.t \mid (t,t)_{\Sigma x:t.t} \mid \pi_1(t) \mid \pi_2(t) \\ & \mid t + t \mid i(t)_{t+t} \mid j(t)_{t+t} \mid \delta(t, x.t, x.t) \mid \mathsf{nat} \mid 0 \mid S \mid R_t \mid =_t \mid L_P \mid \mathsf{refl}_t \mid \bot \mid \mathsf{elim}_\bot(t) \end{aligned}$$

The variable x is bound in the subterm t in $\lambda x : u.t$, $\Pi x : u.t$, $\Sigma x : u.t$, $\delta(u, x.t, y.w)$ and $\delta(u, y.w, x.t)$.

The reductions are:

$$\begin{array}{rcl}
(\lambda x : T.t\ u) & \rhd & t[x \setminus u] \\
\pi_1(t,u)_{\Sigma x:T.U} & \rhd & t \\
\pi_2(t,u)_{\Sigma x:T.U} & \rhd & u \\
\delta(i(t), x.u, y.v) & \rhd & u[x \setminus t] \\
\delta(j(t), x.u, y.v) & \rhd & v[y \setminus t] \\
(R\ 0\ t_0\ t_S) & \rhd & t_0 \\
(R\ S(n)\ t_0\ t_S) & \rhd & (t_S\ n\ (R\ n\ t_0\ t_S)) \\
(L_P\ u\ v\ (\mathsf{refl}_T\ t)\ p) & \rhd & p
\end{array}$$

We write $=_\beta$ for the transitive, relfexive, symmetric contextual closure of $\rhd$.

We do not redo the proof, but Church-Rosser is done in the usual way (albeit with more cases).

Theorem 8.2.1. *If $t =_\beta u$, then there exists a term v such that $t \rhd^* v$ and $u \rhd^* v$.*

In general, the properties of Martin-Löf's type theory are proved in the same way and the same order as for the previous formalisms, especially LF.

8.3 Typing Rules

The rules are all the rules of the previous chapter, extended by the ones given in figure 8.1, which correspond to the additional constructions.

Note that the conversion rule seems unchanged, but actually takes into account the new additional reductions:

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash B : s}{\Gamma \vdash t : B}\ (\text{if } A =_\beta B)$$

8.4 Basic properties

The main properties are, as already stated, similar in form and proof to the ones of LF. We thus do not go into details, but checking some proofs is a good exercise.

Lemma 8.4.1 (Inversion)**.** *If $\Gamma \vdash t : T$, then Γ wf. Furthermore, the conditions of lemma 7.2.5 hold, as well as the following:*

- *If $\Gamma \vdash 0 : T$ then $T =_\beta \mathsf{nat}$ and $\Gamma \vdash T : \mathsf{Type}$.*
- *If $\Gamma \vdash S : T$ then $T =_\beta \mathsf{nat} \to \mathsf{nat}$ and $\Gamma \vdash T : \mathsf{Type}$.*

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma; (x : A) \vdash B : \mathsf{Type}}{\Gamma \vdash \Sigma x : A.B : \mathsf{Type}} \qquad \frac{\Gamma \vdash \Sigma x : A.B : \mathsf{Type} \quad \Gamma \vdash a : A \quad \Gamma \vdash b : B[x \setminus a]}{(a, b)_{\Sigma x:A.B} : \Sigma x : A.B}$$

$$\frac{\Gamma \vdash c : \Sigma x : A.B}{\Gamma \vdash \pi_1(c) : A} \qquad \frac{\Gamma \vdash c : \Sigma x : A.B}{\Gamma \vdash \pi_2(c) : B[x \setminus \pi_1(c)]}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma \vdash B : \mathsf{Type}}{\Gamma \vdash A + B : \mathsf{Type}}$$

$$\frac{\Gamma \vdash A + B : \mathsf{Type} \quad \Gamma \vdash a : A}{\Gamma \vdash i(a)_{A+B} : A + B} \qquad \frac{\Gamma \vdash A + B : \mathsf{Type} \quad \Gamma \vdash b : B}{\Gamma \vdash j(b)_{A+B} : A + B}$$

$$\frac{\Gamma \vdash c : A + B \quad \Gamma \vdash C : \mathsf{Type} \quad \Gamma(x : A) \vdash a : C \quad \Gamma(y : B) \vdash b : C}{\Gamma \vdash \delta(c, x.a, y.b) : C}$$

$$\frac{\Gamma \ \mathsf{wf}}{\Gamma \vdash \mathsf{nat} : \mathsf{Type}} \qquad \frac{\Gamma \ \mathsf{wf}}{\Gamma \vdash 0 : \mathsf{nat}} \qquad \frac{\Gamma \ \mathsf{wf}}{\Gamma \vdash S : \mathsf{nat} \to \mathsf{nat}}$$

$$\frac{\Gamma \vdash P : \mathsf{nat} \to \mathsf{Type}}{\Gamma \vdash R_P : (P\ 0) \to (\Pi n : \mathsf{nat}.(P\ n) \to (P\ (S\ n))) \to \Pi n : \mathsf{nat}.(P\ n)}$$

$$\frac{\Gamma \vdash T : \mathsf{Type}}{\Gamma \vdash =_T : T \to T \to \mathsf{Type}} \qquad \frac{\Gamma \vdash T : \mathsf{Type}}{\Gamma \vdash \mathsf{refl}_T : \Pi x : T.(=_T\ x\ x)}$$

$$\frac{\Gamma \vdash T : \mathsf{Type} \quad \Gamma \vdash P : T \to \mathsf{Type}}{\Gamma \vdash L_P : \Pi x : T.\Pi y : T.(=_T\ x\ y) \to (P\ x) \to (P\ y)}$$

$$\frac{\Gamma \ \mathsf{wf}}{\Gamma \vdash \bot : \mathsf{Type}} \qquad \frac{\Gamma \vdash T : \mathsf{Type}}{\Gamma \vdash \mathsf{elim}_\bot(T) : \bot \to T}$$

Figure 8.1: Additional typing rules for Martin-Löf's Type Theory

- $\Gamma \vdash R_P : T$ *then* $\Gamma \vdash P : \mathsf{nat} \to \mathsf{Type}$ *and* $T =_\beta (P\ 0) \to (\Pi n : \mathsf{nat}.(P\ n) \to (P\ (S\ n))) \to \Pi n : \mathsf{nat}.(P\ n)$.
- *If* $\Gamma \vdash \Sigma x : A.B : T$ *then* $T = \mathsf{Type}$, $\Gamma \vdash A : \mathsf{Type}$ *and* $\Gamma(x : A) \vdash B : \mathsf{Type}$.
- *If* $\Gamma \vdash A + B : T$ *then* $T = \mathsf{Type}$, $\Gamma \vdash A : \mathsf{Type}$ *and* $\Gamma \vdash B : \mathsf{Type}$.
- *If* $\Gamma \vdash (a, b)_{\Sigma x:A.B} : T$ *then* $\Gamma \vdash a : A$, $\Gamma \vdash b : B[x \setminus a]$, $\Gamma \vdash T : \mathsf{Type}$ *and* $T =_\beta \Sigma x : A.B$.
- *If* $\Gamma \vdash \pi_1(t) : T$ *then there exists* A *and* B *such that* $\Gamma \vdash t : \Sigma x : A.B$ *and* $T =_\beta A$.
- *If* $\Gamma \vdash \pi_2(t) : T$ *then there exists* A *and* B *such that* $\Gamma \vdash t : \Sigma x : A.B$ *and* $T =_\beta B[x \setminus \pi_1(t)]$.
- *If* $\Gamma \vdash i(a)_{A+B} : T$ *then* $\Gamma \vdash a : A$ *and* $T =_\beta A + B$.
- *If* $\Gamma \vdash j(b)_{A+B} : T$ *then* $\Gamma \vdash b : B$ *and* $T =_\beta A + B$.
- *If* $\Gamma \vdash \delta(t, x.u, y.v) : T$ *then there exists* A *and* B *such that* $\Gamma \vdash t : A + B$, $\Gamma(x : A) \vdash u : T$ *and* $\Gamma(y : B) \vdash v : T$.
- *If* $\Gamma \vdash =_U : T$ *then* $\Gamma \vdash U : \mathsf{Type}$ *and* $T =_\beta U \to U \to \mathsf{Type}$.
- *If* $\Gamma \vdash \mathsf{refl}_U : T$ $\Gamma \vdash U : \mathsf{Type}$ *and* $T =_\beta \Pi x : U.(=_U\ x\ x)$.
- *If* $\Gamma \vdash L_P : T$ *then there exists* U *such that* $\Gamma \vdash U : \mathsf{Type}$, $\Gamma \vdash P : U \to \mathsf{Type}$ *and* $T =_\beta \Pi x : T.\Pi y : T.(=_T\ x\ y) \to (P\ x) \to (P\ y)$.

Lemma 8.4.2. *If* $\Gamma \vdash t : T$ *and* $\Gamma \vdash t : T'$ *then* $T =_\beta T'$.

Lemma 8.4.3 (Subject reduction)**.** *If* $\Gamma \vdash t : T$ *and* $t \rhd^* t'$ *(resp.* $T \rhd^* T'$*) then* $\Gamma \vdash t' : T$ *(resp.* $\Gamma \vdash t : T'$*).*

Lemma 8.4.4 (Functions in MLTT)**.** *One can build in MLTT a term* t *of type* $\Pi x : A.\Sigma\ y : B.R$ *if and only if one can:*

1. *build a function* $f : A \to B$*, such that*
2. $\Pi x : A.R[y \setminus (f\ x)]$ *is provable.*

Proof. Just take $f \equiv \lambda x : A.\pi_1(t\ x)$. □

8.5 Erasing dependency

Like in LF, we can erase dependency in type. In this case, we map the terms (resp. types) of MLTT to terms (resp. types) of system T (equipped with non-dependent product and sum types). This is useful for two reasons:

- It allows to show strong normalization (SN), by reducing the SN property of MLTT to the one of system T, in the same way as what is described in chapter 7 for LF.

- It allows to show that the functions that are definable in MLTT are the functions that are definable in system T. This can be used to show that these are also the function definable in arithmetic.

We map the types of MLTT to types of system T extended with product and sum types:

$$\begin{array}{rcl}
\overline{\mathsf{nat}} & \equiv & \mathsf{nat} \\
\overline{\Pi x : T.U} & \equiv & \overline{T} \to \overline{U} \\
\overline{\Sigma x : T.U} & \equiv & \overline{T} \times \overline{U} \\
\overline{T + U} & \equiv & \overline{T} + \overline{U} \\
\overline{(T\ t)} & \equiv & \overline{T} \\
\overline{\lambda x : T.U} & \equiv & \overline{U} \\
\overline{X} & \equiv & X \\
\overline{\bot} & \equiv & \mathsf{nat}
\end{array}$$

We map the term of MLTT of type T to terms of type $\overline{T}$:

$$\begin{array}{rcl}
\overline{x} & \equiv & x \\
\overline{(t\ u)} & \equiv & (\bar{t}\ \overline{u}) \\
\overline{0} & \equiv & 0 \\
\overline{S} & \equiv & S \\
\overline{R_T} & \equiv & R_{\overline{T}} \\
\overline{(t,u)_{\Sigma x:A.B}} & \equiv & (\overline{u}, \overline{v}) \\
\ldots &&
\end{array}$$

Lemma 8.5.1. *If $\Gamma \vdash t : T$ in MLTT, then $\overline{\Gamma} \vdash \bar{t} : \overline{T}$.*

Lemma 8.5.2. *If $\Gamma \vdash t : T$ in MLTT and $t \rhd^* t'$ then $\bar{t} \rhd^* \overline{t'}$.*

This entails:

Theorem 8.5.3. *If $[] \vdash t : \mathsf{nat} \to \mathsf{nat}$ in MLTT, then there exists a term of type $\mathsf{nat} \to \mathsf{nat}$ in system T which behaves the same way.*

Corollary 8.5.4. *The functions definable in MLTT are the functions definable in Heyting's arithmetic.*

We can use a slightly more complicated encoding to show that SN for MLTT boils down to SN for system T. The idea is the same than when mapping LF to simple types in the previous chapter; for instance $\overline{\overline{\lambda x : T.t}} = \lambda x : \overline{T}.(\lambda \bar{t}\ \overline{\overline{T}})$. We do not give all the details here. But using the same technique we obtain:

Theorem 8.5.5. *If $\Gamma \vdash t : T$ (in MLTT) then t and T are strongly normalizable.*

Corollary 8.5.6. *Type checking is decidable in MLTT (as presented here).*

Proof. Using lemma reflem:MLTT-inv and strong normalization for checking β-conversion. □

Theorem 8.5.7. *A function is definable in MLTT if and only if it is definable in system T.*

8.6 Constructivity

The constructivity of Type Theory essentially follows from normalization. The technical details depend upon the syntactical details of how the theory is presented. The presentation in this chapter is chosen in order to make it reasonably smooth.

The main point is still that a closed, normal term must be of a form corresponding to an introduction rule, that is a constructor.

Theorem 8.6.1. *If* $[] \vdash t : T$ *is derivable, with* $[] \vdash T : \mathsf{Type}$ *and* t *in normal form, then:*

1. *if* $T =_\beta N$ *then* t *is of the form* $0, S(0), S(S(0)), \ldots$
2. *if* T *reduces to some* $U + V$*, then* t *is of the form* $i(u)$ *of* $j(v)$*,*
3. *if* T *reduces to some* $\Sigma x : U.V$*, then* t *is of the form* (u, v)*,*
4. *if* T *reduces to* $=_U \ a\ b$ *then* t *is of the form* $(\mathsf{refl}_{U'}\ u)$*,*
5. $T = \bot$ *is not possible,*
6. *if* T *reduces to some* $\Pi x : A.B$*, then either:*
   - t *is equal to* S*,*
   - t *is of the form* $\lambda x : A'.u$*,*
   - t *is of the form* $\mathsf{elim}_\bot(U)$*,*
   - t *is of the form* refl_U*,*
   - t *is of one of the forms* L_P*,* $(L_P\ u)$*,* $(L_P\ u\ v)$ *(but not with more arguments applied to* L_P*,*
   - t *is of one of the forms* R_P*,* $(R_P\ p_0)$*,* $(R_P\ p_0\ p_S)$ *(but no third argument to* R_P*).*

Proof. We perform an induction over the structure of t. Going through all the cases is tedious[1] but we go through some cases.

- If t is of the form $\pi_1(u)$, then typing (that is the inversion lemma) ensures that $[] \vdash u : \Sigma x : A.B$ for some A and B. So the induction hypothesis for u ensures that $u = (v, w)$. But then $t = \pi_1(v, w)$ is not normal.

[1] And this is an example illustrating that formal proofs are useful when dealing with theoretical properties of programming languages.

- If t is of the form $\delta(u, x.v, y.w)$ then typing ensures that $[] \vdash u : A + B$ for some A and B. So the induction hypothesis for u ensures that $u = i(u')$ or $u = j(u'')$. But then $t = \delta(i(u'), x.v, y.w)$ or $t = \delta(j(u''), x.v, y.w)$ which are not normal.
- If t is an application the principle is similar but with more cases to consider. One looks at the head term:
  - t cannot be of the form $(x\ t_1\ \ldots\ t_n)$ because it is closed.
  - t can be a partial application of R_P with strictly less than three arguments, in which case it satisfies the condition. If there are at least three arguments, $(R_P\ p_0\ p_S\ u)$ then typing ensures that $[] \vdash u : N$. Then the induction hypothesis ensures that $u = S^i(0)$ for some i. But this means that t is not normal.
  - ...

I encourage you to go through some of the other cases by yourself. □

Corollary 8.6.2. *If $A + B$ is provable without axiom in MLTT, then one can exhibit either an axiom-free proof of A, or an axiom-free proof of B.*

If $\Sigma x : A.B$ is provable without axiom in MLTT, then one can exhibit a closed term $t : A$ and an axiom-free proof of $B[x \setminus t]$.

Chapter 9

System F

I give two versions of system F and write the normalization proof for both. This is a little redundant, there are no very deep differences. The first is often called the Curry style version, and the second the Church style version. The Curry style is more compact, but the terms do not carry enough information for type checking being decidable. The Church version is more in line with the rest of these course notes.

9.1 Curry style

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

$$\textsc{Var}\ \frac{}{\Gamma \vdash x : T}\ (\mathsf{If}\ (x:T) \in \Gamma)$$

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

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

Lemma 9.1.1 (Substitution for terms)**.** *If* $\Gamma(x : U) \vdash t : T$ *and* $\Gamma \vdash u : U$ *then* $\Gamma \vdash t[x \setminus u] : T$.

Proof. By induction over the derivation of $\Gamma(x : U) \vdash t : T$ (and not by induction over t since the two are not isomorphic for this Curry style presentation). □

Lemma 9.1.2 (Substitution for types)**.** *If* $\Gamma \vdash t : T$*, then, for any type* U *and type variable* α*, we have* $\Gamma[\alpha \setminus U] \vdash t : T[\alpha \setminus U]$.

Proof. By induction over the derivation of $\Gamma \vdash t : T$. All cases are straightforward, one just may need to do some type variable renaming for Fa and Inst. □

I state the subject reduction lemma, but it is actually surprisingly difficult to prove for this version of the calculus. The main reason is that the inversion lemma is not straightforward: if $\Gamma \vdash \lambda x.t : U \to T$ it is difficult to show $\Gamma \vdash (x : U) \vdash t : T$. However we have:

Lemma 9.1.3 (Subject reduction)**.** *If $\Gamma \vdash t : T$, and $t \rhd_\beta t'$, then $\Gamma \vdash t' : T$.*

But one actually does not need subject reduction to prove normalization.

Definition 9.1.1. Neutral terms - Curry A term t is said to be neutral ($t \in \mathcal{N}$) if and only if it is not of the form $\lambda x.u$.

Definition 9.1.2 (Reducibility candidate - Curry)**.** A set $\mathcal{C}$ of λ-terms is a reducibility candidate if and only if it verifies the three closure properties:

1. $\mathcal{C} \subset \mathsf{SN}$
2. $\forall t \in \mathcal{A} \cap \mathsf{SN}, t \in \mathcal{C}$
3. if $t \in \mathcal{N}$, and whenever $t \rhd_\beta t'$ one has $t' \in \mathcal{C}$, then $t \in \mathcal{C}$.

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

Lemma 9.1.4. *The set of strongly normalizing terms SN is a reducibility candidate. So $\mathcal{CR}$ is not empty.*

Remark. Any neutral and normal term belongs to any reducibility candiate. In particular all variables x belong to all reducibility candidates. So reducibility candidates are not empty.

Lemma 9.1.5 (Closure CR 1)**.** *If $\mathcal{C}$ and $\mathcal{C}'$ are reducibility candidates, then so is the set $\mathcal{C} \to \mathcal{C}'$ defined by:*

$$\mathcal{C} \to \mathcal{C}' \equiv \{t, \forall u \in \mathcal{C}, (t\ u) \in \mathcal{C}'\}.$$

Lemma 9.1.6 (Closure CR 2)**.** *If $(\mathcal{C}_i)_{i\in I}$ is a (non-empty) family of reducibility candidates, then $\bigcap_{i\in I} \mathcal{C}_i$ is a reducibility candidate.*

Definition 9.1.3 (reducibility sets)**.** Let $\mathcal{I}$ be a mapping from type variables to reducibility candidates. That is $\mathcal{I}(\alpha) \in \mathcal{CR}$ for all $i \in I$. for any type T we define a set $|T|_\mathcal{I}$ of λ-terms by:

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

Following the previous closure lemmas, it is immediate that $|T|_{\mathcal{I}}$ is a reducibility candidate.

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

Lemma 9.1.7. *For all types T and U and type variable α we have*

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

We can then prove the main lemma:

Lemma 9.1.8. *If :*

- $\Gamma \vdash t : T$,
- *$\mathcal{I}$ is a mapping from type variables to reducibility candidates,*
- *σ is a mapping from term variables to terms such that when $(x : U) \in \Gamma$ then $\sigma(x) \in |T|_{\mathcal{I}}$,*

then $t[\sigma] \in |T|_{\mathcal{I}}$.

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

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

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

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

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

□

Lemma 9.1.9. *Suppose that for every term variable x, if $(x : U) \in \Gamma$ then $\sigma(x) \in |U|_{\mathcal{I}}$. Then, when $\Gamma \vdash t : T$ holds, we have $t[\sigma] \in |T|_{\mathcal{I}}$.*

Corollary 9.1.10. *If $\Gamma \vdash t : T$, then $t \in \mathsf{SN}$.*

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

9.1.1 Variants

One can take different definitions for the set of reducibility candidates. Possible variants are:

Definition 9.1.4 (CR, variant 1 (Parigot))**.** The set of reducibility candidates is the smallest set of set of λ-terms such that:

1. SN is a reducibility candidate,
2. if $\mathcal{C}$ and $\mathcal{C}'$ are reducibility candidates, then $\mathcal{C} \to \mathcal{C}'$ is a reducibility candidate,
3. for any family $(\mathcal{C}_i)_{i\in I}$ of reducibility candidates (with I not empty), the intersection $\bigcap_{i\in I} \mathcal{C}_i$ is a reducibility candidate.

Definition 9.1.5 (saturated sets)**.** A set $\mathcal{C}$ is a reducibility candidates when:

1. any strongly normalizing term of the form $(x\ u_1\ u_2\ \dots\ u_n)$ belongs to $\mathcal{C}$,
2. $\forall t \in \mathcal{C}. t \rhd t' \Rightarrow t' \in \mathcal{C}$,
3. if $(t[x \setminus u]\ v_1\ v_2\ \dots\ v_n) \in \mathcal{C}$ and $u \in \mathsf{SN}$, then $(\lambda x.t\ u\ \ v_1\ v_2\ \dots\ v_n) \in \mathcal{C}$.

You can check that these definitions are not strictly equivalent, but they lead to similar proofs.

9.2 Church Style

The set of types is the same as in the previous section. The set of terms becomes:

$$t \quad ::== \quad x \mid \lambda x : T.t \mid (t\ t) \mid \Lambda\alpha.t \mid (t\ T)$$

And the set of typing rules is:

$$\textsc{Var}\ \frac{}{\Gamma \vdash x : T}\ (\mathsf{If}\ (x : T) \in \Gamma)$$

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

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

Lemma 9.2.1 (Inversion)**.**

- *If* $\Gamma \vdash x : T$ *then* $(x : T) \in \Gamma$,
- *if* $\Gamma \vdash \lambda x : U.t : V$ *then* $\Gamma(x : U) \vdash t : T$ *and* $V = U \to T$,
- *if* $\Gamma \vdash (t\ u) : T$ *then* $\Gamma \vdash t : U \to T$ *and* $\Gamma \vdash u : U$,

- *If* $\Gamma \vdash \Lambda\alpha.t : U$ *then* $\Gamma \vdash t : T$, $U = \forall\alpha.T$ *and* α *is not free in* Γ,
- *if* $\Gamma \vdash (t\ U) : V$ *then* $\Gamma \vdash t : \forall\alpha.T$ *and* $V = T[\alpha \setminus U]$.

Corollary 9.2.2. *If* $\Gamma \vdash t : T$ *and* $\Gamma \vdash t : T'$, *then* $T = T'$ *(or more precisely* $T =_\alpha T'$*).*

The proof of subject reduction is then in line with what is show in the previous chapters for other type systems.

We can use the same definition(s) of reducibility candidates as in the previous section for Curry terms. We just have to adjust the definition of neutral terms:

Definition 9.2.1 (Neutral terms)**.** A term is neutral, if it is not of one of the following forms: $\lambda x : T.t$, $\Lambda\alpha.t$.

The definition of reducibility sets becomes:

Definition 9.2.2. Given a mapping $\mathcal{I}$ from type variables to reducibility candidates, we define for every type T a set $|T|_{\mathcal{I}}$ of λ-terms by:

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

The main lemma becomes:

Lemma 9.2.3. *If :*

- $\Gamma \vdash t : T$,
- $\mathcal{I}$ *is a mapping from type variables to reducibility candidates,*
- σ *is a mapping from term variables to terms such that when* $(x : U) \in \Gamma$ *then* $\sigma(x) \in |T|_{\mathcal{I}}$,
- θ *is a mapping from type variables to types,*

then $t[\sigma][\theta] \in |T|_{\mathcal{I}}$.

Proof. The proof is by induction over the structure of t (or equivalently over the derivation of $\Gamma \vdash t : T$).

Let us look at the "new" cases:

- If $t = \Lambda\alpha.u$, we know $T = \forall\alpha.U$ and $\Gamma \vdash u : U$ with α not occurring in Γ.

  we have to show that for all type V any reducibility candidate $\mathcal{C}$ and substitutions σ and θ, $((\Lambda.\alpha.u)[\sigma][\theta]\ V) \in |U|_{\mathcal{I};\alpha\leftarrow\mathcal{C}}$.

  Since the $|U|_{\mathcal{I};\alpha\leftarrow\mathcal{C}}$ is a reducibility candidate, it is enough to show that any reduct of $((\Lambda.\alpha.u)[\sigma][\theta]\ V))$ is in the set. This is done by induction over the length of the longest reduction path starting with u. The key case is the head redex, that is showing that: $u[\sigma][\theta; \alpha \setminus V] \in |U|_{\mathcal{I};\alpha\leftarrow\mathcal{C}}$. This is precisely ensured by the induction hypothesis.

- If $t = (u\ V)$, we know that $T = U[\alpha \setminus V]$ and $\Gamma \vdash u : \forall\alpha.U$.

  We have to show that $(u\ V)[\sigma][\theta] \in |U[\alpha \setminus V]|_{\mathcal{I}}$, which is equivalent to $(u[\sigma][\theta]\ V[\theta]) \in |U[\alpha \setminus V]|_{\mathcal{I}}$ and thus to $(u[\sigma][\theta]\ V[\theta]) \in= |U|_{\mathcal{I};\alpha\leftarrow|V|_{\mathcal{I}}}$.

  Furthermore, the induction hypothesis ensures that $u[\sigma][\theta] \in |\forall\alpha.U|_{\mathcal{I}}$. By definition of $|\forall\alpha.U|_{\mathcal{I}}$, this immediately entails the result.

□

Corollary 9.2.4. *There is no closed term of type $\forall\alpha.\alpha$ in system F.*

9.3 Encoding data in System F

Although its definition is very concise, System F is very expressive.

Definition 9.3.1 (Cartesian product)**.** If A and B are types, we can define the product in system F by:

$$\begin{aligned} A \times B &\equiv \forall\alpha.(A \to B \to \alpha) \to \alpha \\ \\ \mathit{pair} &: A \to B \to A \times B \\ &\equiv \lambda a : A.\lambda b : B.\Lambda\alpha.\lambda f : A \to B \to \alpha.(f\ a\ b) \\ \\ \pi_1 &: A \times B \to A \\ &\equiv \lambda c : A \times B.(c\ A\ \lambda a : A.\lambda b : B.a) \\ \\ \pi_2 &: A \times B \to B \\ &\equiv \lambda c : A \times B.(c\ A\ \lambda a : A.\lambda b : B.B) \end{aligned}$$

Definition 9.3.2 (Sum type)**.** If A and B are types, we can define the sum in system F by:

$$\begin{aligned} A + B &\equiv \forall\alpha.(A \to \alpha) \to (B \to \alpha) \to \alpha \\ \\ i &: A \to A + B \\ &\equiv \lambda a : A.\Lambda\alpha.\lambda f : A \to \alpha.\lambda g : B \to \alpha.(f\ a) \\ \\ j &: B \to A + B \\ &\equiv \lambda b : B.\Lambda\alpha.\lambda f : A \to \alpha.\lambda g : B \to \alpha.(g\ b) \end{aligned}$$

Definition 9.3.3 (Church numerals)**.**

$$\begin{aligned} N &\equiv \forall\alpha.\alpha \to (\alpha \to \alpha) \to \alpha \\ \\ 0 &\equiv \lambda\alpha.\lambda x : \alpha.\lambda f : \alpha \to \alpha.x \\ S &\equiv \lambda n : N.\lambda\alpha.\lambda x : \alpha.\lambda f : \alpha \to \alpha.(n\ (f\ x)\ f) \end{aligned}$$

Chapter 10

System F, à la Curry

10.1 Syntax

Terms, types, and contexts are given by the following grammars.

$$
\begin{aligned}
t &::= x \mid (t\,t) \mid \lambda x.t && \text{(terms)}\\
T &::= \alpha \mid T \to T \mid \forall \alpha.T && \text{(types)}\\
\Gamma &::= [\,] \mid \Gamma(x : T) && \text{(contexts)}
\end{aligned}
$$

Note that, à la Curry, abstraction $\lambda x.t$ carries no type annotation: terms are exactly the raw terms of the untyped λ-calculus. Polymorphism is introduced and eliminated purely at the level of typing derivations, via the rules FA and INST below; no term-former corresponding to $\Lambda\alpha.t$ or $t[T]$ is needed.

10.2 Typing rules

Typing judgments have the form $\Gamma \vdash t : T$. There are five rules: VAR, APP, LAM, FA (generalization), INST (instantiation).

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T}\ (\textsc{var}) \qquad \frac{\Gamma \vdash t : T \to U \quad \Gamma \vdash u : T}{\Gamma \vdash (t\,u) : U}\ (\textsc{app})$$

$$\frac{\Gamma(x : T) \vdash t : U}{\Gamma \vdash \lambda x.t : T \to U}\ (\textsc{lam})$$

$$\frac{\Gamma \vdash t : T \quad \alpha \notin \mathrm{FV}(\Gamma)}{\Gamma \vdash t : \forall \alpha.T}\ (\textsc{fa}) \qquad \frac{\Gamma \vdash t : \forall \alpha.T}{\Gamma \vdash t : T[\alpha := U]}\ (\textsc{inst})$$

Remark. Because FA and INST act only on the type side of the judgment, the same raw term t may be assigned many different types. In particular, there is no separate "erasure" map to define: the PER model built below lives directly on the term syntax itself.

10.3 Reduction and strong normalization

We do not dwell on the technicalities of binding and substitution (de Bruijn indices vs. named variables): terms are considered up to α-equivalence, and $t[x := u]$ denotes the usual capture-avoiding substitution.

One-step β-reduction $\triangleright$ is the compatible closure of

$$(\lambda x.t)\, u \;\triangleright\; t[x := u].$$

Definition 10.3.1 (Strong normalization)**.** A term t is *strongly normalizing*, written $t \in \mathsf{SN}$, if every reduction sequence issued from t is finite.

10.4 Saturated PERs

Let Λ denote the set of raw terms defined in Section 10.1.

Definition 10.4.1 (Saturated PER)**.** A *saturated PER* is a relation $R \subseteq \Lambda \times \Lambda$ such that:

(1) R is symmetric and transitive;

(2) if $t\,R\,u$ then $t, u \in \mathsf{SN}$;

(3) if $t\,R\,u$ and $t \triangleright t'$, then $t'\,R\,u$;

(4) if n, m are neutral and $n, m \in \mathsf{SN}$, then $n\,R\,m$;

(5) if $(t[x := u]\, u_1 \cdots u_n)\,R\,v$ and $u \in \mathsf{SN}$, then $(\lambda x.t)\, u\, u_1 \cdots u_n\, R\, v$.

A term is *neutral* if it is a variable applied to zero or more arguments, $x\, u_1 \cdots u_n$.

Lemma 10.4.1 (SN of extended applications)**.** *If* $t[x := u]\, u_1 \cdots u_n \in \mathsf{SN}$ *and* $u \in \mathsf{SN}$*, then* $(\lambda x.t)\, u\, u_1 \cdots u_n \in \mathsf{SN}$*.*

Proof. From $t[x := u]\, u_1 \cdots u_n \in \mathsf{SN}$: each $u_i \in \mathsf{SN}$ and $t[x := u] \in \mathsf{SN}$ (subterms of an SN application); by a simulation argument (reduction commutes with substitution, so an infinite reduction of t would yield one of $t[x := u]$), $t \in \mathsf{SN}$, hence $\lambda x.t \in \mathsf{SN}$. Any infinite reduction sequence from $(\lambda x.t)\, u\, u_1 \cdots u_n$ either eventually contracts the head redex $(\lambda x.t)\, u$ – landing in $t[x := u]\, u_1 \cdots u_n$, shown SN, hence only finitely many further steps – or never does, staying confined to the already- SN parts $\lambda x.t$, u, $u_1, \ldots, u_n$; both are impossible. □

Remark. Clause (5), as stated, only requires $u \in \mathsf{SN}$: the seemingly stronger requirement that the whole application $(\lambda x.t)\, u\, u_1 \cdots u_n$ be SN is in fact equivalent, given the other clauses. From $(t[x := u]\, u_1 \cdots u_n)\, R\, v$, CR1 already gives $t[x := u]\, u_1 \cdots u_n \in \mathsf{SN}$, so Lemma 10.4.1 recovers $(\lambda x.t)\, u\, u_1 \cdots u_n \in \mathsf{SN}$ from the weaker hypothesis $u \in \mathsf{SN}$ – and conversely $u \in \mathsf{SN}$ is immediate from the stronger hypothesis (a subterm of an SN application is SN).

Remark. Clauses (2)–(5) are the relational counterparts of the usual reducibility-candidate conditions: (2) is CR1, (3) is CR2, (5) is the expansion clause for β-redexes (guarded by the SN-ness of u, so as not to admit terms such as $(\lambda x.y)\,\Omega$), and (4) is the clause for neutral terms, in the style of Altenkirch's saturated Λ-sets: *any* two neutral SN terms are related, regardless of their arguments. Semantically, this amounts to interpreting every open (variable-headed) term by a single generic value $\bot$. This is coarser than relating only matching spines with pointwise-related arguments, but it is harmless for the purposes of strong normalization (see the discussion in Chapter 11).

10.5 Closure of saturated PERs

Definition 10.5.1 (Arrow relation)**.** Given $R, Q \subseteq \Lambda \times \Lambda$, define $R \to Q$ by

$$t\ (R \to Q)\ t' \quad \Longleftrightarrow \quad \forall u, u'.\ u\,R\,u' \Rightarrow (t\,u)\,Q\,(t'\,u').$$

Theorem 10.5.1. *If R and Q are saturated PERs, then so is $R \to Q$.*

Proof. (1) Symmetry. Suppose $t\,(R \to Q)\,t'$; let $u\,R\,u'$. By symmetry of R, $u'\,R\,u$, so $(t\,u')\,Q\,(t'\,u)$; by symmetry of Q, $(t'\,u)\,Q\,(t\,u')$. Hence $t'\,(R \to Q)\,t$.

Transitivity. Suppose $t\,(R \to Q)\,t'$ and $t'\,(R \to Q)\,t''$; let $u\,R\,u'$. Since R is a PER, $u'\,R\,u'$. Thus $(t\,u)\,Q\,(t'\,u')$ and $(t'\,u')\,Q\,(t''\,u')$, so by transitivity of Q, $(t\,u)\,Q\,(t''\,u')$.

(2) CR1. Suppose $t\,(R \to Q)\,t'$. Since a variable x is neutral and trivially SN, clause (4) gives $x\,R\,x$; applying the hypothesis, $(t\,x)\,Q\,(t'\,x)$, so $t\,x \in \mathsf{SN}$ by CR1 of Q, hence $t \in \mathsf{SN}$ (a subterm of an SN application is SN). Likewise $t' \in \mathsf{SN}$.

(3) CR2. Suppose $t\,(R \to Q)\,t'$ and $t \triangleright t''$; let $u\,R\,u'$. Then $(t\,u)\,Q\,(t'\,u')$ and $t\,u \triangleright t''\,u$, so by CR2 of Q, $(t''\,u)\,Q\,(t'\,u')$.

(4) Neutral terms. Suppose n, m are neutral and $n, m \in \mathsf{SN}$. We show $n\,(R \to Q)\,m$: let $a\,R\,a'$. By CR1 of R, $a, a' \in \mathsf{SN}$; since n, m are neutral, $n\,a$ and $m\,a'$ are neutral, and SN (a neutral term applied to an SN argument is SN). By clause (4) of Q, applied to $n\,a$ and $m\,a'$, $(n\,a)\,Q\,(m\,a')$.

(5) Expansion. Suppose $\big(t[x := u]\,u_1 \cdots u_n\big)\,Q\,v$ and $u \in \mathsf{SN}$. We show $(\lambda x.t)\,u\,u_1 \cdots u_n\,(R \to Q)\,v$: let $a\,R\,a'$. Applying the first hypothesis to (a, a'): $\big(t[x := u]\,u_1 \cdots u_n\,a\big)\,Q\,(v\,a')$. Since $u \in \mathsf{SN}$, clause (5) of Q – applied with argument list $u_1, \ldots, u_n, a$ – gives $\big((\lambda x.t)\,u\,u_1 \cdots u_n\,a\big)\,Q\,(v\,a')$, as required. □

Theorem 10.5.2. *Let $(R_i)_{i \in I}$ be a non-empty family of saturated PERs. Then $\bigcap_{i \in I} R_i$ is a saturated PER.*

Proof. Write $S = \bigcap_{i \in I} R_i$; fix some $j \in I$ (possible since $I \neq \emptyset$). Symmetry and transitivity of S follow pointwise from those of each R_i. If $t\,S\,u$ then $t\,R_j\,u$, so $t, u \in \mathsf{SN}$ by CR1 of R_j. If $t\,S\,u$ and $t \triangleright t'$, then $t\,R_i\,u$ for every i, so $t'\,R_i\,u$ for every i (CR2 of each R_i), hence $t'\,S\,u$. Clauses (4) and (5) are likewise checked componentwise: the hypotheses of each clause, read against S, hold against every R_i, so the conclusions hold against every R_i, hence against S. □

10.6 Interpretation of types

Notation 2. *Let* SAT *denote the set of saturated PERs on* Λ.

Definition 10.6.1 (Interpretation of types)**.** Let $\mathcal{I}$ be a map from type variables to SAT. The interpretation $[\![A]\!]_{\mathcal{I}}$ of a type A is defined by induction on the structure of A:

$$\begin{aligned} [\![\alpha]\!]_{\mathcal{I}} &= \mathcal{I}(\alpha) \\ [\![A \to B]\!]_{\mathcal{I}} &= [\![A]\!]_{\mathcal{I}} \to [\![B]\!]_{\mathcal{I}} \\ [\![\forall \alpha.A]\!]_{\mathcal{I}} &= \bigcap_{C \in \mathrm{SAT}} [\![A]\!]_{\mathcal{I}[\alpha \mapsto C]} \end{aligned}$$

By Theorems 10.5.1 and 10.5.2 (the intersection defining $[\![\forall \alpha.A]\!]_{\mathcal{I}}$ being over the non-empty set SAT), $[\![A]\!]_{\mathcal{I}} \in \mathrm{SAT}$ for every type A.

Lemma 10.6.1 (Substitution for Interpretation)**.** *For any types T, U and type variable α, we have:*

$$[\![T[\alpha \setminus U]]\!]_{\mathcal{I}} = [\![T]\!]_{\mathcal{I};\alpha \leftarrow [\![U]\!]_{\mathcal{I}}}$$

Proof. By induction over the structure of T. □

10.7 Soundness and strong normalization

Definition 10.7.1 (Compatible environments)**.** A *term environment* is a map η from term variables to Λ; for a raw term t, $\eta(t)$ denotes t with its free variables replaced according to η. Given a context Γ and $\mathcal{I}$ as above, two environments η, η' are Γ*-linked (w.r.t. $\mathcal{I}$)* if, for every $x : T \in \Gamma$, $\eta(x)\ [\![T]\!]_{\mathcal{I}}\ \eta'(x)$.

Theorem 10.7.1 (Soundness)**.** *If $\Gamma \vdash t : T$, then for every $\mathcal{I}$ and every pair of environments η, η' that are Γ-linked w.r.t. $\mathcal{I}$,*

$$\eta(t)\ [\![T]\!]_{\mathcal{I}}\ \eta'(t).$$

Proof. We proceed by induction over the derivation of $\Gamma \vdash t : T$, looking at each typing rule.

VAR The judgement is $\Gamma \vdash x : T$ with $x : T \in \Gamma$. So $\eta(x)\ [\![T]\!]_{\mathcal{I}}\ \eta'(x)$ holds, which is precisely what we want.

APP The judgement is $\Gamma \vdash (t\,u) : U$. The induction hypotheses ensure that: $\eta(t)\ [\![U \to T]\!]_{\mathcal{I}}\ \eta'(t)$ and $\eta(u)\ [\![U]\!]_{\mathcal{I}}\ \eta'(u)$. Which by definition of substitution and of $[\![U \to T]\!]_{\mathcal{I}}$ ensures $\eta(t\,u)\ [\![U]\!]_{\mathcal{I}}\ \eta'(t\,u)$.

LAM The judgement is $\Gamma \vdash \lambda x.t : T \to U$ and the induction hypothesis is for $\Gamma(x : T) \vdash t : U$. Given $u\ [\![T]\!]_{\mathcal{I}}\ u'$ we need to prove

$$(\eta(\lambda x.t)\,u)\ [\![U]\!]_{\mathcal{I}}\ (\eta'(\lambda x.t)\,u').$$

Defining θ (resp. θ') as the application identical to η (resp. η') everywhere except for x which it maps to itself, this is identical to:

$$(\lambda x.\theta(t)\, u)\ [\![U]\!]_{\mathcal{I}}\ (\lambda x.\theta'(t)\, u').$$

Since $u\ [\![T]\!]_{\mathcal{I}}\ u'$, CR1 gives $u, u' \in \mathsf{SN}$. By the induction hypothesis (applied below), together with clause (5) of $[\![U]\!]_{\mathcal{I}}$, it suffices to show:

$$(1)\quad \theta(t)[x \setminus u]\ [\![U]\!]_{\mathcal{I}}\ \theta'(t)[x \setminus u'].$$

But $\theta(t)[x \setminus u]$ is t substituted by $\eta; x \leftarrow u$. Furthermore $\eta; x \leftarrow u$ is Γ-linked to $\eta'; x \leftarrow u'$. So we have (1) by the induction hypothesis. It remains to derive the goal from (1): since $u \in \mathsf{SN}$, clause (5) applied to (1) gives $(\lambda x.\theta(t))\, u\ [\![U]\!]_{\mathcal{I}}\ \theta'(t)[x \setminus u']$; by symmetry, $\theta'(t)[x \setminus u']\ [\![U]\!]_{\mathcal{I}}\ (\lambda x.\theta(t))\, u$, and since $u' \in \mathsf{SN}$, clause (5) applied again gives $(\lambda x.\theta'(t))\, u'\ [\![U]\!]_{\mathcal{I}}\ (\lambda x.\theta(t))\, u$; by symmetry once more, $(\lambda x.\theta(t))\, u\ [\![U]\!]_{\mathcal{I}}\ (\lambda x.\theta'(t))\, u'$, as required.

FA The judgement is $\Gamma \vdash t : \forall\alpha.T$ and the premiss $\Gamma \vdash t : T$. Because α is not free in Γ, the property of being Γ-*linked w.r.t.* $\mathcal{I}$ does not depend upon the value $\mathcal{I}(\alpha)$. So whatever the value chosen for $\mathcal{I}(\alpha)$, if η, η' are Γ-linked w.r.t. $\mathcal{I}$, the induction hypothesis ensures

$$\eta(t)\ [\![T]\!]_{\mathcal{I}}\ \eta'(t).$$

INST The judgement is $\Gamma \vdash t : T[\alpha \setminus U]$ and the premiss $\Gamma \vdash t : \forall\alpha.T$. Given η and η', we must show

$$\eta(t)\ [\![T[\alpha \setminus U]]\!]_{\mathcal{I}}\ \eta'(t).$$

Because of lemma 10.6.1, this is equivalent to

$$\eta(t)\ [\![T]\!]_{\mathcal{I};\alpha \leftarrow [\![U]\!]_{\mathcal{I}}}\ \eta'(t)$$

which is precisely (one instance of) the induction hypothesis.

□

Corollary 10.7.2 (Strong normalization of System F)**.** *If* $\Gamma \vdash t : T$*, then* $t \in \mathsf{SN}$.

Proof. Let $\eta = \eta'$ be the identity environment ($\eta(x) = x$ for every variable x). For $x : T' \in \Gamma$, x is neutral and trivially SN, so $x\ [\![T']\!]_{\mathcal{I}}\ x$ by clause (4) of Definition 10.4.1, for any $\mathcal{I}$; hence η, η' are Γ-linked w.r.t. any $\mathcal{I}$. By Theorem 10.7.1, $t = \eta(t)\ [\![T]\!]_{\mathcal{I}}\ \eta'(t) = t$, i.e. $t\ [\![T]\!]_{\mathcal{I}}\ t$. By CR1 (clause (2) of Definition 10.4.1), $t \in \mathsf{SN}$. □

Chapter 11

The full system: CC + Nat + rec + R

11.1 The system

We follow, for this dependently-typed system, a two-judgment presentation (context formation Γ wf and typing $\Gamma \vdash e : T$, defined by mutual induction) rather than the more compact single-judgment style used for the PTS core in some textbooks.

11.1.1 Sorts

The system has two sorts,

$$\mathcal{S} = \{\mathsf{Set}, \mathsf{Type}\},$$

one axiom, and four product rules:

$$\mathcal{A} = \{(\mathsf{Set}, \mathsf{Type})\} \qquad \mathcal{R} = \{(\mathsf{Set}, \mathsf{Set}),\ (\mathsf{Type}, \mathsf{Set}),\ (\mathsf{Set}, \mathsf{Type}),\ (\mathsf{Type}, \mathsf{Type})\}.$$

There is no axiom $\mathsf{Type} : \mathsf{Type}$ and no sort above Type: the system has no universe hierarchy.

11.1.2 Expressions and contexts

As is customary for PTS-style systems, types and terms are not syntactically distinguished; both are *expressions*:

$$e ::= s \mid x \mid \Pi x{:}e.\, e \mid \lambda x{:}e.\, e \mid (e\, e) \mid \mathsf{Nat} \mid \mathsf{0} \mid \mathsf{succ}\, e \mid \mathsf{rec}\, e\, e\, e \mid \mathsf{R}\, e\, e\, e$$

where s ranges over $\mathcal{S}$. Contexts are, as before, $\Gamma ::= [\,] \mid \Gamma(x : e)$.

Notation 3. *We write $T \to U$ for $\Pi x{:}T.\, U$ when $x \notin \mathrm{FV}(U)$.*

11.1.3 Reduction

We reuse the notation $\triangleright$, $\triangleright^*$, $\sim_\beta$ from Chapter 1, now for the compatible closure (resp. reflexive-transitive closure, resp. the generated equivalence) of β-reduction together with the following ι-rules:

$$
\begin{aligned}
(\lambda x{:}T.\, t)\, u &\triangleright t[x := u] && (\beta)\\
\mathsf{rec}\, z\, f\, \mathsf{0} &\triangleright z && (\iota_1)\\
\mathsf{rec}\, z\, f\, (\mathsf{succ}\, n) &\triangleright f\, n\, (\mathsf{rec}\, z\, f\, n) && (\iota_2)\\
\mathsf{R}\, T_0\, T_S\, \mathsf{0} &\triangleright T_0 && (\iota_3)\\
\mathsf{R}\, T_0\, T_S\, (\mathsf{succ}\, n) &\triangleright T_S\, n\, (\mathsf{R}\, T_0\, T_S\, n) && (\iota_4)
\end{aligned}
$$

Remark. Confluence of $\triangleright$ – needed, in particular, for rule CONV below to be well-behaved (e.g. for types to have essentially unique sorts) – is a metatheoretic property to be established separately; we take it for granted here, as is standard for systems of this shape.

11.1.4 Judgments

Context formation.

$$\frac{}{[\,]\ \mathrm{wf}}\ (\textsc{wf-emp}) \qquad \frac{\Gamma \vdash T : s \quad x \notin \mathrm{dom}(\Gamma)}{\Gamma(x : T)\ \mathrm{wf}}\ (\textsc{wf-ext})$$

Core typing.

$$\frac{\Gamma\ \mathrm{wf}}{\Gamma \vdash \mathsf{Set} : \mathsf{Type}}\ (\textsc{ax}) \qquad \frac{\Gamma\ \mathrm{wf} \quad x : T \in \Gamma}{\Gamma \vdash x : T}\ (\textsc{var})$$

$$\frac{\Gamma \vdash T : s_1 \quad \Gamma(x : T) \vdash U : s_2 \quad (s_1, s_2) \in \mathcal{R}}{\Gamma \vdash \Pi x{:}T.\, U : s_2}\ (\textsc{prod})$$

$$\frac{\Gamma(x : T) \vdash t : U \quad \Gamma \vdash \Pi x{:}T.\, U : s}{\Gamma \vdash \lambda x{:}T.\, t : \Pi x{:}T.\, U}\ (\textsc{abs})$$

$$\frac{\Gamma \vdash t : \Pi x{:}T.\, U \quad \Gamma \vdash u : T}{\Gamma \vdash (t\, u) : U[x := u]}\ (\textsc{app}) \qquad \frac{\Gamma \vdash t : T \quad \Gamma \vdash T' : s \quad T \sim_\beta T'}{\Gamma \vdash t : T'}\ (\textsc{conv})$$

Naturals.

$$\frac{\Gamma\ \mathrm{wf}}{\Gamma \vdash \mathsf{Nat} : \mathsf{Set}}\ (\textsc{nat}) \qquad \frac{\Gamma\ \mathrm{wf}}{\Gamma \vdash \mathsf{0} : \mathsf{Nat}}\ (\textsc{zero}) \qquad \frac{\Gamma \vdash n : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}\, n : \mathsf{Nat}}\ (\textsc{succ})$$

Recursors.

$$\frac{\Gamma \vdash T : \mathsf{Nat} \to \mathsf{Set} \quad \Gamma \vdash z : T\, \mathsf{0} \quad \Gamma \vdash f : \Pi k{:}\mathsf{Nat}.\, T\, k \to T\, (\mathsf{succ}\, k) \quad \Gamma \vdash n : \mathsf{Nat}}{\Gamma \vdash \mathsf{rec}\, z\, f\, n : T\, n}\ (\textsc{rec})$$

$$\frac{\Gamma \vdash T_0 : \mathsf{Set} \quad \Gamma \vdash T_S : \mathsf{Nat} \to \mathsf{Set} \to \mathsf{Set} \quad \Gamma \vdash n : \mathsf{Nat}}{\Gamma \vdash \mathsf{R}\, T_0\, T_S\, n : \mathsf{Set}}\ (\mathrm{R})$$

Remark. Rule REC is now the fully dependent (Curry–Howard / induction principle) eliminator for Nat, not the plain System-T iterator: the motive T is itself Nat-indexed, $T : \mathsf{Nat} \to \mathsf{Set}$, and the step function f produces, at each k, an element of $T\,(\mathsf{succ}\,k)$ out of one of $T\,k$. One can check the ι-rules of Section 11.1 remain well-typed: $z : T\,\mathsf{0}$ matches $\mathsf{rec}\,z\,f\,\mathsf{0} : T\,\mathsf{0}$, and $f\,n\,(\mathsf{rec}\,z\,f\,n) : T\,(\mathsf{succ}\,n)$ matches $\mathsf{rec}\,z\,f\,(\mathsf{succ}\,n) : T\,(\mathsf{succ}\,n)$.

Remark. This brings REC and R structurally closer than before: both are now driven by a Nat-indexed family (a genuine motive $T : \mathsf{Nat} \to \mathsf{Set}$ for REC, the pair (T_0, T_S) encoding one for R). Consequently REC's semantics will need the same treatment as R's for the case where n is neutral (Section 11.5): $[\![\mathsf{rec}\,z\,f\,n]\!]_{\mathcal{I}}$, like $[\![\mathsf{R}\,T_0\,T_S\,n]\!]_{\mathcal{I}}$, cannot in general be computed by unfolding the $\mathbb{N}$-indexed fold at a non-numeral n, and will need its own "generic" treatment there.

Remark (Open point, for discussion). Should APP also carry an explicit premise $\Gamma \vdash T : s$ – redundant here, since T already occurs as the domain of a well-typed Π-type via rule PROD, but some presentations state it anyway for uniformity with ABS? I have left it out, as the leaner choice.

11.2 Extending the base algebra and saturated PERs

11.3 Closure by Π and impredicative quantification

11.4 Soundness theorem (fundamental lemma)

11.5 Nat, rec, and R

11.6 Discussion: classical logic and parametricity

Bibliography

[1] H. Barendregt. Lambda calculi with types. Technical Report 91-19, Catholic University Nijmegen, 1991. in Handbook of Logic in Computer Science, Vol II.

[2] H.P. Barendregt. *The Lambda Calculus its Syntax and Semantics*. North-Holland, 1981.

[3] E. Bishop. *Foundations of Constructive Analysis*. McGraw-Hill, 1967.

[4] C. Böhm and A. Berarducci. Automatic synthesis of typed λ-programs on term algebras. *Theoretical Computer Science*, 39, 1985.

[5] R. S. Boyer and J. S. Moore. *A computational logic*. ACM Monograph. Academic Press, 1979.

[6] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5 (1), pp. 56-68, 1940.

[7] A. Church. *The Calculi of Lambda-Conversion*. Princeton University Press, 1941.

[8] Th. Coquand and G. Huet. Constructions: A higher order proof system for mechanizing mathematics. In *EUROCAL'85*, volume 203 of *Lecture Notes in Computer Science*, Linz, 1985. Springer-Verlag.

[9] Th. Coquand and G. Huet. Concepts mathématiques et informatiques formalisés dans le calcul des constructions. In The Paris Logic Group, editor, *Logic Colloquium'85*. North-Holland, 1987.

[10] Th. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76(2/3), 1988.

[11] Th. Coquand and C. Paulin-Mohring. Inductively defined types. In P. Martin-Löf and G. Mints, editors, *Proceedings of Colog'88*. Springer-Verlag, 1990. LNCS 417.

[12] N.J. De Bruijn. Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indag. Math.*, 34 (5), pp. 381–392, 1972.

[13] N.J. De Bruijn. A survey of the project automath. In J.P. Seldin and J.R. Hindley, editors, *to H.B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press, 1980.

[14] E. W. Dijkstra, editor. *Formal Development of Programs and Proofs*. The UT Year of Programming Series. Addison-Wesley, 1989.

[15] A. Doxiadis, C.H. Papadimitriou, A. Papadatos, and A. Di Donna. *Logicomix: An Epic Search for Truth*. Bloomsbury USA, 2009.

[16] R. Cori et D. Lascar. *Logique Mathématique, Cours et exercices*. Axiomes. Masson, 1993.

[17] G. Gentzen. *The Collected Work of Gerherd Gentzen*. North-Holland, 1969. Édité par M.E. Szabo.

[18] E. Giménez. A tutorial on recursive types in coq. Rapport Technique 0221, INRIA, May 1998.

[19] J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la téorie des types. In *Proceedings of the 2nd Scandinavian Logic Symposium*. North-Holland, 1970.

[20] J.-Y. Girard. Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur, 1972.

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

[22] K. Gödel. Über einer bisher noch nicht benützte erweiterung des finiten standpunktes. *Dialectica*, 12, 1958. Traduit dans le Journal of Philosophical Logic 9, 1980. Réedité avec traduction anglaise dans les œuvres complètes.

[23] A. Heyting. *Intuitionism, an introduction*. Studies in Logic and the foundations of Mathematics. North-Holland, 1966.

[24] J. Roger Hindley and Jonathan P. Seldin. *Lambda-Calculus and Combinators: An Introduction*. Cambridge University Press, Cambridge, 2nd edition, 2008.

[25] C.A.R. Hoare, editor. *Concurrent Programming*. The UT Year of Programming Series. Addison-Wesley, 1989.

[26] W.A. Howard. The formulae-as-types notion of constructions. In J.P. Seldin and J.R. Hindley, editors, *to H.B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1980. Unpublished 1969 Manuscript.

[27] G. Huet. A uniform approach to type theory. In G. Huet, editor, *Logical Foundations of Logical Programming*, pages 337–397. Addison–Weseley, 1990. Rapport de Recherche 795, INRIA, 1988.

[28] G. Huet. Residual theory in λ-calculus: A complete gallina development. Technical Report 2002, INRIA, 1993.

[29] S.C. Kleene. *Introduction to Metamathematics*. Bibliotheca Mathematica. North-Holland, 1952.

[30] S.C. Kleene. *Introduction to Metamathematics*. Bibliotheca Mathematica. North-Holland, 1952.

[31] J.-L. Krivine. *Théorie Axiomatique des Ensembles.* Presses Universitaires de France, 1969.

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

[33] P. Martin-Löf. A theory of types. Technical report, Report 71-3, Dept. of Mathematics, Université de Stockholm, 1971.

[34] P. Martin-Löf. *Intuitionistic Type Theory.* Studies in Proof Theory. Bibliopolis, 1984.

[35] F. Honsell et G. Plotkin R Harper. A framework for defining logics. *Journal of the ACM*, 40 (1), pp. 143–184, 1993.

[36] L. Thery, Y. Bertot, and G. Kahn. Real theorem provers deserve real user-interfaces. Rapport de Recherche 1684, INRIA Sophia, May 1992.

[37] D. T. van Daalen. *The Language Theory of Automath.* PhD thesis, Technische Hogeschool Eindhoven, 1980.

[38] 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.