Chapter 2

Proof Reflection in Coq

We formalise natural deduction for first-order logic in the proof assistant Coq, using De Bruijn indices for variable binding. The main judgement we model is of the form $\Gamma \vdash d$ [:] ϕ, stating that d is a proof term of formula ϕ under hypotheses Γ; it can be viewed as a typing relation by the Curry–Howard–De Bruijn isomorphism. This relation is proved sound with respect to Coq's native logic and is amenable to the manipulation of formulas and of derivations. As an illustration, we define a reduction relation on proof terms with permutative conversions and prove the property of subject reduction.

Author: Dimitri Hendriks

2.1 Introduction

We represent intuitionistic predicate logic in Coq [66], an interactive proof construction system that implements the calculus of inductive constructions [69], which is a type theory that provides inductive definitions. We adopt a two-level approach [8] in the sense that the native logic of the system is the meta-language in which we define and reason about our object-language. The object-language consists of a deep embedding of first-order terms, formulas and derivation terms. Derivation terms and formulas are related on the meta-level by definition of a deduction system for hypothetical judgements $\Gamma \vdash d$ [:] ϕ, that encapsulate their own evidence; d *inhabits* ϕ given context Γ. Several binding mechanisms are handled by De Bruijn indices [19].

The main contribution of our work is that we design an object language representing first-order logic, which can be used as a 'tool' for the manipulation of formulas and proofs. Moreover, via the so-called *reflection* operation [18] and the soundness result, it's possible to reason about the first-order fragment of the native logic itself.

The meta-theory of lambda calculi, type and proof systems of various kinds has already been treated quite extensively with the use of theorem provers, as shown, for example, in [50, 2, 10, 40, 6, 51, 7].

To our knowledge, this is the first complete (first-order) formalisation of natural deduction for first-order logic using analytic judgements. Although we realise that our work is just standard first-order logic, and the results proved are the first basic ones, the strong point is the achievement of a Coq implementation, a library which can be reused in the future for several purposes, of which we mention:

- investigating the meta-theory of deduction systems, and
- proving correctness of proof-search algorithms.

Finally, we think that our work might serve as an overview on how to formalise logical systems in a theorem prover.

The complete development is formalised in Coq and can be retrieved from [33]; it's size is 116184 bytes, 4980 lines. The development time is approximately half a man-year.

For a brief introduction to type theory and the Coq proof assistant, an explanation of reflection and the two-level approach, and the motivation of our design choices with respect to variable binding mechanisms and the format of hypothetical judgements, the reader is referred to the preface.

This chapter is organised as follows. In Section 2.2 we introduce objects representing first-order terms, first-order formulas and derivation terms. In Sections 2.3 and 2.4, we define lifting and substitution. In Section 2.5 some basic algebraic properties of the defined De Bruijn operations are listed. The inference rules for hypothetical judgements are presented in Section 2.6. In Section 2.7 we show that the structural rules are admissible. In Section 2.8 we define the translation from object level formulas to their meta-level counterparts. In Section 2.9 we discuss an alternative set-up with a finite number of free variables instead of infinitely many; and we discuss an inconvenient aspect of substitution of free variables. Section 2.10 presents thinning and substitution lemmas about this translation function, necessary for the proof of soundness with respect to Coq's logic, given in Section 2.11. In Section 2.12, we specify a function which infers the type of (correct) proof terms. In Section 2.13 this function is proved correct with respect to the outlined inference system. As a corollary, derivation terms have unique types (Section 2.14). Section 2.15 serves as an example of how the defined machinery can be used to manipulate/transform proof terms; Prawitz's proof reduction rules are defined. In Section 2.16 we present soundness of types for the defined proof reduction, the property known as subject reduction. Finally, we conclude and discuss future work.

2.2 Objects

A logic is usually defined with respect to a signature determining its sorts, function symbols and predicate symbols. In our formalisation of intuitionistic predicate logic, we choose to deal with one sort only. We freed ourselves of the technical care multiple sorts would demand, simply for practical reasons.[1]

[1]It is well-known that sorts can be built-in artificially by using unary predicates.

The sets τ (terms), o (formulas) and π (proof terms), defined in the present section, depend on the signature—constituted by two arbitrary but fixed lists of natural numbers, representing function and relation arities. This dependence remains implicit in the sequel. We motivate this design choice.

A first (set-theoretical) attempt to formalise the dependency of an arbitrary signature would be to depart from an abstract set of function symbols, say F, along with an abstract function, say $\mathsf{arity} : F \to \mathbb{N}$. Given our aim to gain full control over the object language, however, this is unsatisfactory in several respects, of which we mention

- the undecidability of equality of terms, and

- the impossibility to check whether a function symbol occurs in a term.

Admittedly, one can add the necessary axioms. For example, we can assume the existence of a Boolean predicate $\mathsf{eqb} : F \to F \to \mathsf{bool}$. Of course, we then have to show consistency, but this doesn't seem to be problematic. What matters is the conceptual difference. With the approach chosen here, signatures are first class citizens and are finite, as opposed to the representation with F, arity and eqb. Such a representation of functions is what we called in the introduction a shallow embedding where the interpretation function of object-level function symbols to meta-level function symbols is the identity. As said before, the disadvantage of a shallow embedding is the impossibility to exploit the syntactical structure.[2]

Instead, we use an index set for function (as well as for relation) symbols.

Definition 2.2.1 *Given a set A, lists of type $\mathsf{list}(A)$ are defined by $\Box$ and $[a|l]$ where $a : A$ and $l : \mathsf{list}(A)$. Given a list $l : \mathsf{list}(A)$, its* index set *I_l is defined by the equations:*

$$I_\Box = \emptyset \qquad I_{[a|l']} = \mathbf{1} + I_{l'}$$

where $\emptyset$ is the empty set (i.e., without contructors), $\mathbf{1}$ the unit set (with sole inhabitant $\bullet$) and $A + B$ the disjoint sum of sets A and B, defined inductively by:

$$A + B := \mathsf{inl}(a) \mid \mathsf{inr}(b)$$

where $a : A$ and $b : B$. The application $l(i)$ computes the list element indexed by $i : I_l$, as defined by the following recursion:

$$\begin{array}{rcl} [a|l](\mathsf{inl}(\bullet)) & = & a \\ [a|l](\mathsf{inr}(i)) & = & l(i) \end{array}$$

For the sake of readability we set $I_l = \{0, \ldots, |l| - 1\}$, where $|l|$ denotes the length of l.

[2]A philosopher might raise his finger and swap things around: "A shallow embedding of objects in combination with full control over those objects, leads to well-known classical complications, such as diagonalisation, paradoxes and worse."

Definition 2.2.2 (Terms) *Assume a list of natural numbers, representing function arities.*

$$l_{\mathsf{fun}} : \mathsf{list}(\mathbb{N})$$

The set τ of syntactic objects representing first-order terms *is inductively defined by:*

$$\tau := v_n \mid f_i(t_1, \dots, t_k)$$

where $n : \mathbb{N}$, $i : I_{l_{\mathsf{fun}}}$, $k = l_{\mathsf{fun}}(i)$ and $t_1, \dots, t_k : \tau$. It is to be understood that $l_{\mathsf{fun}}(i)$ computes the arity k of f_i (if $k = 0$, then $f_i()$ is a constant).

Definition 2.2.3 (Formulas) *We assume a second list of natural numbers, representing relation arities.*

$$l_{\mathsf{rel}} : \mathsf{list}(\mathbb{N})$$

The set of objects o representing predicate logical formulas, *is defined by the following abstract syntax, where $j : I_{l_{\mathsf{rel}}}$, $m = l_{\mathsf{rel}}(j)$ and $\phi, \chi : o$.*

$$o := \dot{\top} \mid \dot{\bot} \mid R_j(t_1, \dots, t_m) \mid \phi \mathrel{\dot{\rightarrow}} \chi \mid \phi \mathbin{\dot{\wedge}} \chi \mid \phi \mathbin{\dot{\vee}} \chi \mid \dot{\forall}\, \phi \mid \dot{\exists}\, \phi$$

As usual, we write $\dot{\neg}\phi$ as shorthand for $\phi \mathrel{\dot{\rightarrow}} \dot{\bot}$.

We use the following binding priorities for the connectives: $\dot{\forall}, \dot{\exists} > \dot{\wedge}, \dot{\vee} > \dot{\rightarrow}$ and let binary connectives associate to the right. For example, $\dot{\exists}\, \phi \mathbin{\dot{\vee}} \dot{\exists}\, \chi \mathrel{\dot{\rightarrow}} \dot{\exists}\, (\phi \mathbin{\dot{\vee}} \chi)$ reads as $((\dot{\exists}\, \phi) \mathbin{\dot{\vee}} (\dot{\exists}\, \chi)) \mathrel{\dot{\rightarrow}} \dot{\exists}\, (\phi \mathbin{\dot{\vee}} \chi)$.

In the sequel, when we write $f_i(t_1, \dots, t_k)$ or $R_j(t_1, \dots, t_m)$, we implicitly assume:

$$i : I_{l_{\mathsf{fun}}} \quad l_{\mathsf{fun}}(i) = k \qquad j : I_{l_{\mathsf{rel}}} \quad l_{\mathsf{rel}}(j) = m$$

We now turn to the definition of derivation terms, which can be seen as linear notations for two-dimensional proof trees.

Definition 2.2.4 (Derivations) *The syntactic class π of* proof terms *is defined by the grammar:*

$$\begin{array}{rcl} \pi & := & \top^+ \mid h_n \mid \bot^-(d, \phi) \mid \rightarrow^+(\phi, d) \mid \rightarrow^-(d, e) \\ & & \mid \wedge^+(d, e) \mid \wedge_l^-(d) \mid \wedge_r^-(d) \mid \vee_l^+(\phi, d) \mid \vee_r^+(\phi, d) \mid \vee^-(d, e, f) \\ & & \mid \forall^+(d) \mid \forall^-(t, d) \mid \exists^+(\phi, t, d) \mid \exists^-(d, e) \end{array}$$

where $n : \mathbb{N}$, $d, e, f : \pi$, $\phi : o$ and $t : \tau$. Note that the h_n are assumption variables, as will become clear in the sequel.

As an example, we depict the construction $\vee^-(d, e_1, e_2)$ in traditional natural deduction format:

$$\dfrac{\begin{array}{c} \\ \vdots\ (d) \\ \chi_1 \mathbin{\dot{\vee}} \chi_2 \end{array} \quad \begin{array}{c} [\chi_1] \\ \vdots\ (e_1) \\ \phi \end{array} \quad \begin{array}{c} [\chi_2] \\ \vdots\ (e_2) \\ \phi \end{array}}{\phi}\ \vee^-$$

Some constructors ($\bot^-$, $\to^+$, $\vee_l^+$, $\vee_r^+$ and $\exists^+$) carry an argument of type o in order to have proof terms uniquely determine natural deductions, as will be shown in the sequel (see Section 2.12). Had we omitted the formula argument in, for example, $\to^+$, a term $\to^+(h_0)$ would be ambiguous in the sense that it serves as a proof term for $\phi \dot{\to} \phi$ for any $\phi : o$. Thus, we use explicit Church style typing. The formula argument in $\exists^+$ is required, because there is no inverse of substitution, that is, we cannot deduce ϕ from $\phi[t]$ (see Definition 2.6.1).

2.3 Recursive Patterns

Several object (of types o and π) transformations concerning (assumption as well as term) variables recursively descend in the same way. These recursive patterns are shared by abstracting from what should happen to terms or assumption variables.

The operations carry an argument storing the so-called *reference depth* of variables, because variables can only be 'grasped' (lifted, substituted, etc.) if we know at what reference depth they reside.

For objects in o, the reference depth increments when a quantifier is passed.

Definition 2.3.1 *Given $n : \mathbb{N}$, $g : \mathbb{N} \to \tau \to \tau$ and $\phi : o$, define $\mathsf{map}_o(g, n, \phi)$ as follows.*

$$\begin{array}{rcl}
\mathsf{map}_o(g,n,c) & = & c \textit{ for } c = \dot{\top}, \dot{\bot} \\
\mathsf{map}_o(g,n,R_j(t_1,\ldots,t_m)) & = & R_j(g(n,t_1),\ldots,g(n,t_m)) \\
\mathsf{map}_o(g,n,\phi \circ \chi) & = & \mathsf{map}_o(g,n,\phi) \circ \mathsf{map}_o(g,n,\chi) \textit{ for } \circ = \dot{\to}, \dot{\wedge}, \dot{\vee} \\
\mathsf{map}_o(g,n,\mathcal{Q}\phi) & = & \mathcal{Q}\mathsf{map}_o(g,n+1,\phi) \textit{ for } \mathcal{Q} = \dot{\forall}, \dot{\exists}
\end{array}$$

For proof terms, the reference depth of *term* variables v_i increments in the cases of $\forall^+$, $\exists^+$ (first argument) and $\exists^-$ (second argument).

Definition 2.3.2 *Given $g : \mathbb{N} \to \tau \to \tau$, $n : \mathbb{N}$ and $d : \pi$, the function $\mathsf{map}^{\mathsf{v}}_\pi(g, n, d)$ is defined by the following recursive equations.*

$$\begin{array}{rcl}
\mathsf{map}^{\mathsf{v}}_\pi(g,n,\top^+) & = & \top^+ \\
\mathsf{map}^{\mathsf{v}}_\pi(g,n,\bot^-(d,\phi)) & = & \bot^-(\mathsf{map}^{\mathsf{v}}_\pi(g,n,d),\mathsf{map}_o(g,n,\phi)) \\
\mathsf{map}^{\mathsf{v}}_\pi(g,n,h_i) & = & h_i \\
\mathsf{map}^{\mathsf{v}}_\pi(g,n,\to^+(\phi,d)) & = & \to^+(\mathsf{map}_o(g,n,\phi)),\mathsf{map}^{\mathsf{v}}_\pi(g,n,d)) \\
\mathsf{map}^{\mathsf{v}}_\pi(g,n,\to^-(d,e)) & = & \to^-(\mathsf{map}^{\mathsf{v}}_\pi(g,n,d),\mathsf{map}^{\mathsf{v}}_\pi(g,n,e)) \\
\mathsf{map}^{\mathsf{v}}_\pi(g,n,\wedge^+(d,e)) & = & \wedge^+(\mathsf{map}^{\mathsf{v}}_\pi(g,n,d),\mathsf{map}^{\mathsf{v}}_\pi(g,n,e)) \\
\mathsf{map}^{\mathsf{v}}_\pi(g,n,\wedge_l^-(d)) & = & \wedge_l^-(\mathsf{map}^{\mathsf{v}}_\pi(g,n,d)) \\
\mathsf{map}^{\mathsf{v}}_\pi(g,n,\wedge_r^-(d)) & = & \wedge_r^-(\mathsf{map}^{\mathsf{v}}_\pi(g,n,d)) \\
\mathsf{map}^{\mathsf{v}}_\pi(g,n,\vee_l^+(\phi,d)) & = & \vee_l^+(\mathsf{map}_o(g,n,\phi)),\mathsf{map}^{\mathsf{v}}_\pi(g,n,d)) \\
\mathsf{map}^{\mathsf{v}}_\pi(g,n,\vee_r^+(\phi,d)) & = & \vee_r^+(\mathsf{map}_o(g,n,\phi)),\mathsf{map}^{\mathsf{v}}_\pi(g,n,d))
\end{array}$$

$$\begin{array}{rcl}
\mathsf{map}^{\mathsf{v}}_{\pi}(g,n,\vee^-(d,e_1,e_2)) & = & \vee^-(\mathsf{map}^{\mathsf{v}}_{\pi}(g,n,d),\mathsf{map}^{\mathsf{v}}_{\pi}(g,n,e_1),\mathsf{map}^{\mathsf{v}}_{\pi}(g,n,e_2)) \\
\mathsf{map}^{\mathsf{v}}_{\pi}(g,n,\forall^+(d)) & = & \forall^+(\mathsf{map}^{\mathsf{v}}_{\pi}(g,n+1,d)) \\
\mathsf{map}^{\mathsf{v}}_{\pi}(g,n,\forall^-(t,d)) & = & \forall^-(g(n,t),\mathsf{map}^{\mathsf{v}}_{\pi}(g,n,d)) \\
\mathsf{map}^{\mathsf{v}}_{\pi}(g,n,\exists^+(\phi,t,d)) & = & \exists^+(\mathsf{map}_o(g,n+1,\phi)),g(n,t),\mathsf{map}^{\mathsf{v}}_{\pi}(g,n,d)) \\
\mathsf{map}^{\mathsf{v}}_{\pi}(g,n,\exists^-(d,e)) & = & \exists^-(\mathsf{map}^{\mathsf{v}}_{\pi}(g,n,d),\mathsf{map}^{\mathsf{v}}_{\pi}(g,n+1,e))
\end{array}$$

Note the increment of the reference depth of the formula argument in $\exists^+$. Consider the inference rule corresponding to $\exists^+$ given in Definition 2.6.1. The argument ϕ in term $\exists^+(\phi,t,d)$ has free variable v_0 ('from the outside'), for which the witnessing t is substituted in the type $\phi[t]$ of subterm d. This free variable should remain free; therefore the reference depth is incremented.

Also, the recursive pattern for proof term transformations concerning *assumption* variables will be reused several times in the sequel. The reference depth of assumption variables h_i is incremented in the cases of $\to^+$ (second argument), $\vee^-$ (second and third argument) and $\exists^-$ (second argument); that is, any time an extra hypothesis is added to the context (the inference rules of Definition 2.6.1 viewed bottom up).

Definition 2.3.3 *Let $g : \mathbb{N} \to \mathbb{N} \to \pi$, a function that returns a proof term given two natural numbers (reference depth, resp. index of assumption variable), $n : \mathbb{N}$, and $d : \pi$, then $\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,d)$ is defined by the following recursive equations.*

$$\begin{array}{rcl}
\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,\top^+) & = & \top^+ \\
\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,\bot^-(d,\phi)) & = & \bot^-(\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,d),\phi) \\
\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,h_i) & = & g(n,i) \\
\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,\to^+(\phi,d)) & = & \to^+(\phi,\mathsf{map}^{\mathsf{h}}_{\pi}(g,n+1,d)) \\
\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,\to^-(d,e)) & = & \to^-(\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,d),\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,e)) \\
\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,\wedge^+(d,e)) & = & \wedge^+(\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,d),\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,e)) \\
\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,\wedge^-_l(d)) & = & \wedge^-_l(\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,d)) \\
\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,\wedge^-_r(d)) & = & \wedge^-_r(\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,d)) \\
\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,\vee^+_l(\phi,d)) & = & \vee^+_l(\phi,\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,d)) \\
\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,\vee^+_r(\phi,d)) & = & \vee^+_r(\phi,\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,d)) \\
\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,\vee^-(d,e_1,e_2)) & = & \vee^-(\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,d),\mathsf{map}^{\mathsf{h}}_{\pi}(g,n+1,e_1), \\
 & & \quad \mathsf{map}^{\mathsf{h}}_{\pi}(g,n+1,e_2)) \\
\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,\forall^+(d)) & = & \forall^+(\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,d)) \\
\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,\forall^-(t,d)) & = & \forall^-(t,\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,d)) \\
\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,\exists^+(\phi,t,d)) & = & \exists^+(\phi,t,\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,d)) \\
\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,\exists^-(d,e)) & = & \exists^-(\mathsf{map}^{\mathsf{h}}_{\pi}(g,n,d),\mathsf{map}^{\mathsf{h}}_{\pi}(g,n+1,e))
\end{array}$$

2.4 Lifting and Substitution

The representation of variables by De Bruijn indices requires an extra operation called *lifting*.[3] Lifting increments the *free* variables in a formula.

We start with defining the operations of lifting and substitution, using side conditions. The implementation uses computationally more efficient definitions, as listed thereafter.

Definition 2.4.1 *We define* term lifting $\upharpoonleft_n t$ *by structural recursion on* $t : \tau$*, where* $n : \mathbb{N}$ *is the reference depth. The first* n *variables,* $v_0, \ldots, v_{n-1}$*, are assumed to be bound (this information being imported from functions calling* $\upharpoonleft_n t$*) and remain unchanged.*

$$\begin{array}{rcl} \upharpoonleft_n v_i & = & \begin{cases} v_i & \textit{if } i < n \\ v_{i+1} & \textit{if } i \geq n \end{cases} \\ \upharpoonleft_n f_i(t_1, \ldots, t_k) & = & f_i(\upharpoonleft_n t_1, \ldots, \upharpoonleft_n t_k) \end{array}$$

We write $\upharpoonleft t$ *to denote the lifting of all variables in* t*, shorthand for* $\upharpoonleft_0 t$*.*

Definition 2.4.2 Substitution of t' for v_n in t, *notation* $t[t']^n$*, is defined by recursion on the structure of* t*. Again,* n *is the reference depth, present in order to deal with substitution under binders. Thus, the first* n *variables should remain untouched. The term* t' *is lifted such that capture by binders is avoided. Indices greater than* n *are decremented, because substitution removes the original variable* v_n*.*

$$\begin{array}{rcl} v_i[t]^n & = & \begin{cases} v_i & \textit{if } i < n \\ \upharpoonleft_0^n t & \textit{if } i = n \\ v_{i-1} & \textit{if } i > n \end{cases} \\ f_i(t_1, \ldots, t_k)[t]^n & = & f_i(t_1[t]^n, \ldots, t_k[t]^n) \end{array}$$

where $\upharpoonleft_n^m t$ *is defined by* $\upharpoonleft_n^0 t = t$ *and* $\upharpoonleft_n^{m+1} t = \upharpoonleft_n^m(\upharpoonleft_n t)$*. We set* $t[t'] = t[t']^0$*.*

As mentioned, the side-conditions (if $i < n$, etc.) in the above definitions are inefficient. As the unfolding of definitions proceeds, the number of side-conditions increases exponentially. The implemented lifting and substitution functions are defined recursively and have no side-conditions $\upharpoonleft_n t$ is encoded as $\mathsf{lift\_trm}(n, t)$ and $t[t']^n$ is encoded as $\mathsf{subst\_trm}(n, t, t')$.[4]

$$\begin{array}{rcl} \mathsf{lift}(0, i) & = & i + 1 \\ \mathsf{lift}(n + 1, 0) & = & 0 \\ \mathsf{lift}(n + 1, i + 1) & = & \mathsf{lift}(n, i) + 1 \end{array}$$

[3]In the literature on explicit substitutions (e.g., [9]) the operation we call *lifting* here consists of two more primitive operations: *lifting* $\Uparrow$ of substitutions and the *shift* substitution $\uparrow$, which increments the indices in a term. Our $\upharpoonleft_n$ actually corresponds to $\Uparrow^n(\uparrow)$.

[4]We found these definitions in [57], where they are attributed to [2].

$$\begin{array}{rcl} \mathsf{lift\_trm}(n, v_i) & = & v_{\mathsf{lift}(n,i)} \\ \mathsf{lift\_trm}(n, f_j(t_1, \ldots, t_k)) & = & f_j(\mathsf{lift\_trm}(n, t_1), \ldots, \mathsf{lift\_trm}(n, t_k)) \end{array}$$

$$\begin{array}{rcl} \mathsf{subst}(0, 0, t) & = & t \\ \mathsf{subst}(0, i+1, t) & = & v_i \\ \mathsf{subst}(n+1, 0, t) & = & v_0 \\ \mathsf{subst}(n+1, i+1, t) & = & \mathsf{lift\_trm}(0, \mathsf{subst}(n, i, t)) \end{array}$$

$$\begin{array}{rcl} \mathsf{subst\_trm}(n, v_i, t) & = & \mathsf{subst}(n, i, t) \\ \mathsf{subst\_trm}(n, f_j(t_1, \ldots, t_k), t) & = & f_j(\mathsf{subst\_trm}(n, t_1, t), \\ & & \quad \ldots, \mathsf{subst\_trm}(n, t_k, t)) \end{array}$$

Next we define the lifting and substitution operations on formulas.

Definition 2.4.3 The lifting of $\phi : o$ for reference depth n, *notation* $\uparrow_n \phi$, *is defined as follows.*

$$\uparrow_n \phi = \mathsf{map}_o(\lambda m : \mathbb{N}.\, \lambda t : \tau.\, \uparrow_m t, n, \phi)$$

Let $\uparrow \phi$ *abbreviate* $\uparrow_0 \phi$, *the increment of all* free *variables in* ϕ.

Definition 2.4.4 Substitution of $t : \tau$ for v_n[5] in $\phi : o$, *notation* $\phi[t]^n$, *is defined as follows.*

$$\phi[t]^n = \mathsf{map}_o(\lambda m : \mathbb{N}.\, \lambda u : \tau.\, u[t]^m, n, \phi)$$

For the inference system introduced in the next section, we also need the lifting and substitution operation on *contexts*. Contexts are defined by $\square$ and $\Gamma; \phi$, where $\phi : o$ and Γ is a context.

Definition 2.4.5 Lifting of all free variables in context Γ, *given that the first* n *variables are bound, notation* $\uparrow_n \Gamma$, *is defined by:*

$$\begin{array}{rcl} \uparrow_n \square & = & \square \\ \uparrow_n (\Gamma; \phi) & = & \uparrow_n \Gamma; \uparrow_n \phi \end{array}$$

Substitution of t for v_n in Γ, *written* $\Gamma[t]^n$, *is defined by:*

$$\begin{array}{rcl} \square[t]^n & = & \square \\ (\Gamma; \phi)[t]^n & = & \Gamma[t]^n; \phi[t]^n \end{array}$$

Again, we write $\uparrow \Gamma$ *for* $\uparrow_0 \Gamma$ *and* $\Gamma[t]$ *for* $\Gamma[t]^0$.

The type checking function, introduced in Section 2.12, requires the definition of the inverse of lifting: *projection*.

[5]The $n+1$-th free variable 'as seen from the outside'.

Definition 2.4.6 *We define* term projection, $\downarrow_n t$*, as follows.*

$$\begin{array}{rcl} \downarrow_n v_i & = & \begin{cases} v_i & \text{if } i \leq n \\ v_{i-1} & \text{if } i > n \end{cases} \\ \downarrow_n f_i(t_1, \ldots, t_k) & = & f_i(\downarrow_n t_1, \ldots, \downarrow_n t_k) \end{array}$$

Formula projection, $\downarrow_n \phi$*, is defined as follows.*

$$\downarrow_n \phi = \mathsf{map}_o(\lambda m{:}\mathbb{N}.\, \lambda t{:}\tau.\, \downarrow_m t, n, \phi)$$

Define $\downarrow \phi = \downarrow_0 \phi$.

Lemma 2.4.1 *For all* $n : \mathbb{N}$ *and* $\phi : o$*, we have that* $\downarrow_n \uparrow_n \phi = \phi$.

Also needed for Definition 2.12.1 is the ability to check whether a variable occurs free in a formula.

Definition 2.4.7 $v_n \in \mathsf{FV}(\phi)$ *is defined as follows.*[6]

$$\begin{array}{ll} v_n \in \mathsf{FV}(R_j(t_1, \ldots, t_m)) & \text{if } v_n \in t_i \text{ for some } 1 \leq i \leq m \\ v_n \in \mathsf{FV}(\phi \circ \chi) & \text{if } v_n \in \mathsf{FV}(\phi) \text{ or } v_n \in \mathsf{FV}(\chi) \text{ for } \circ = \dot{\rightarrow}, \dot{\wedge}, \dot{\vee} \\ v_n \in \mathsf{FV}(\mathcal{Q}\phi) & \text{if } v_{n+1} \in \mathsf{FV}(\phi) \text{ for } \mathcal{Q} = \dot{\forall}, \dot{\exists} \end{array}$$

with $v_n \in t$ *defined by:*

$$\begin{array}{ll} v_n \in v_m & \text{if } n = m \\ v_n \in f_i(t_1, \ldots, t_k) & \text{if } v_n \in t_j \text{ for some } 1 \leq j \leq k \end{array}$$

Lemma 2.4.2 *For all* $n : \mathbb{N}$ *and* $\phi : o$*, we have that* $\uparrow_n \downarrow_n \phi = \phi$*, if* $v_n \notin \mathsf{FV}(\phi)$.

Lemma 2.4.3 *For all* $n : \mathbb{N}$*,* $t : o$*, we have* $v_n \notin \mathsf{FV}(\uparrow_n t)$.

Definition 2.4.8 *For* $n : \mathbb{N}$ *and* $d : \pi$*,* lifting of term variables in proof terms $\uparrow^{\mathsf{v}}_n d$ *is defined as follows.*

$$\uparrow^{\mathsf{v}}_n d = \mathsf{map}^{\mathsf{v}}_\pi(\lambda m{:}\mathbb{N}.\, \lambda t{:}\tau.\, \uparrow_m t, n, d)$$

Definition 2.4.9 *For* $n : \mathbb{N}$*,* $t : \tau$ *and* $d : \pi$*,* substitution of term variables in proof terms $d[t]^n_{\mathsf{v}}$ *is defined by*

$$d[t]^n_{\mathsf{v}} = \mathsf{map}^{\mathsf{v}}_\pi(\lambda m{:}\mathbb{N}.\, \lambda u{:}\tau.\, u[t]^m, n, d)$$

Definition 2.4.10 Lifting of assumption variables in proof terms *is defined by*

$$\uparrow^{\mathsf{h}}_n d = \mathsf{map}^{\mathsf{h}}_\pi(\lambda m{:}\mathbb{N}.\, \lambda i{:}\mathbb{N}.\, h_{\mathsf{lift}(m,i)}, n, d)$$

The function lift *is defined on page 29. Define* $\uparrow^{\mathsf{h}} d = \uparrow^{\mathsf{h}}_0 d$.

[6]We present $v_n \in \mathsf{FV}(\phi)$ as an inductive relation; it's implementation actually is a Boolean function.

Definition 2.4.11 Substitution of proof terms for assumption variables *is defined by*

$$
\begin{array}{rcl}
\top^+[d']^n_{\mathsf{h}} & = & \top^+ \\
h_i[d']^n_{\mathsf{h}} & = & \begin{cases} h_i & \textit{if } i < n \\ \uparrow^n_0 d' & \textit{if } i = n \\ h_{i-1} & \textit{if } i > n \end{cases} \\
\bot^-(d,\phi)[d']^n_{\mathsf{h}} & = & \bot^-(d[d']^n_{\mathsf{h}},\phi) \\
\rightarrow^+(\phi,d)[d']^n_{\mathsf{h}} & = & \rightarrow^+(\phi,d[d']^{n+1}_{\mathsf{h}}) \\
\rightarrow^-(d,e)[d']^n_{\mathsf{h}} & = & \rightarrow^-(d[d']^n_{\mathsf{h}},e[d']^n_{\mathsf{h}}) \\
\wedge^+(d,e)[d']^n_{\mathsf{h}} & = & \wedge^+(d[d']^n_{\mathsf{h}},e[d']^n_{\mathsf{h}}) \\
\wedge^-_l(d)[d']^n_{\mathsf{h}} & = & \wedge^-_l(d[d']^n_{\mathsf{h}}) \\
\wedge^-_r(d)[d']^n_{\mathsf{h}} & = & \wedge^-_r(d[d']^n_{\mathsf{h}}) \\
\vee^+_l(\phi,d)[d']^n_{\mathsf{h}} & = & \vee^+_l(\phi,d[d']^n_{\mathsf{h}}) \\
\vee^+_r(\phi,d)[d']^n_{\mathsf{h}} & = & \vee^+_r(\phi,d[d']^n_{\mathsf{h}}) \\
\vee^-(d,e_1,e_2)[d']^n_{\mathsf{h}} & = & \vee^-(d[d']^n_{\mathsf{h}},e_1[d']^{n+1}_{\mathsf{h}},e_2[d']^{n+1}_{\mathsf{h}}) \\
\forall^+(d)[d']^n_{\mathsf{h}} & = & \forall^+(d[\uparrow^{\mathsf{v}} d']^n_{\mathsf{h}}) \\
\forall^-(t,d)[d']^n_{\mathsf{h}} & = & \forall^-(t,d[d']^n_{\mathsf{h}}) \\
\exists^+(\phi,t,d)[d']^n_{\mathsf{h}} & = & \exists^+(\phi,t,d[d']^n_{\mathsf{h}}) \\
\exists^-(d,e)[d']^n_{\mathsf{h}} & = & \exists^-(d[d']^n_{\mathsf{h}},e[\uparrow^{\mathsf{v}} d']^{n+1}_{\mathsf{h}})
\end{array}
$$

where $\uparrow^m_n d$ *is defined by* $\uparrow^0_n d = d$ *and* $\uparrow^{m+1}_n d = \uparrow^m_n(\uparrow^{\mathsf{h}}_n d)$. *It should be noted that* $h_i[d']^n_{\mathsf{h}}$ *is encoded without side-conditions, in a similar way as* $v_i[t]^n$ *(see Definition 2.4.2). Define* $d[d']_{\mathsf{h}} = d[d']^0_{\mathsf{h}}$.

Note that it is *not* correct to define $d[d']^n_{\mathsf{h}}$ by $\mathsf{map}^{\mathsf{h}}_{\pi}(\lambda m{:}\mathbb{N}.\,\lambda i{:}\mathbb{N}.\,h_i[d']^m_{\mathsf{h}},n,d)$[7], because all free variables v_i in d' have to be lifted to avoid capture of the first free variable by $\forall^+$ or $\exists^-$ (second argument).

2.5 Properties of De Bruijn Operations

We present some basic algebraic properties of the operations introduced in Section 2.4. Similar properties can be found in [7].[8] All five lemmas are proved for both $t : \tau$ as well as for $t : o$.[9] Furthermore $t', t_1, t_2 : \tau$ and $n, m : \mathbb{N}$.

Lemma 2.5.1 (Permutation of lifting)

$$\uparrow_m(\uparrow_n t) = \uparrow_{n+1}(\uparrow_m t) \ \textit{if } m \leq n$$

[7]As we did in [32] (though not in the Coq development).

[8]Where they are attributed to [40].

[9]Similar properties have been proved for lifting and substitution in proof terms, but these are not used in the sequel.

Lemma 2.5.2 (Simplification of substitution)

$$(\uparrow_n t)[t']^n = t$$

Lemma 2.5.3 (Commutation of lifting and substitution)

$$\uparrow_m(t[t']^n) = (\uparrow_m t)[t']^{n+1} \ \textit{if}\ m \leq n$$

Lemma 2.5.4 (Distribution of lifting over substitution)

$$\uparrow_{m+k}(t[t']^m) = (\uparrow_{m+k+1} t)[\uparrow_k t']^m$$

Lemma 2.5.5 (Distribution of substitution)

$$(t[t_1]^m)[t_2]^{m+k} = (t[t_2]^{m+k+1})[t_1[t_2]^k]^m$$

2.6 Judgements

We introduce *judgements* of the form $\Gamma \vdash d\ [:]\ \phi$, stating that d is a proof term of formula ϕ under hypotheses Γ.[10] Alternatively, the object d can be seen as a λ-*term of type* ϕ given variables h_i of type $\Gamma(i)$ for $0 \leq i < |\Gamma|$.

Definition 2.6.1 *The relation* $(\Gamma \vdash d\ [:]\ \phi) : *^p$ *is inductively defined by the following clauses. A context* Γ *is a list of formulas, where the rightmost element has index* 0*,* d, d_1, d_2, e_1, e_2 *are proof terms,* t *is a first-order term, and* $\phi, \phi_1, \phi_2, \chi$ *are formulas.*

$$\frac{}{\Gamma; \phi \vdash h_0\ [:]\ \phi} \qquad \frac{\Gamma \vdash h_i\ [:]\ \chi}{\Gamma; \phi \vdash h_{i+1}\ [:]\ \chi}$$

$$\frac{}{\Gamma \vdash \top^+\ [:]\ \dot{\top}} \qquad \frac{\Gamma \vdash d\ [:]\ \dot{\bot}}{\Gamma \vdash \bot^-(d, \phi)\ [:]\ \phi}$$

$$\frac{\Gamma; \phi \vdash d\ [:]\ \chi}{\Gamma \vdash \to^+(\phi, d)\ [:]\ \phi \dot{\to} \chi} \qquad \frac{\Gamma \vdash d\ [:]\ \phi \dot{\to} \chi \quad \Gamma \vdash e\ [:]\ \phi}{\Gamma \vdash \to^-(d, e)\ [:]\ \chi}$$

$$\frac{\Gamma \vdash d_1\ [:]\ \phi_1 \quad \Gamma \vdash d_2\ [:]\ \phi_2}{\Gamma \vdash \wedge^+(d_1, d_2)\ [:]\ \phi_1 \dot{\wedge} \phi_2}$$

$$\frac{\Gamma \vdash d\ [:]\ \phi_1 \dot{\wedge} \phi_2}{\Gamma \vdash \wedge_l^-(d)\ [:]\ \phi_1} \qquad \frac{\Gamma \vdash d\ [:]\ \phi_1 \dot{\wedge} \phi_2}{\Gamma \vdash \wedge_r^-(d)\ [:]\ \phi_2}$$

[10]We use this notation in order to distinguish '[:]' from ':', which is reserved for the typing relation of Coq.

$$\frac{\Gamma \vdash d \;[:]\; \phi_1}{\Gamma \vdash \vee_l^+(\phi_2, d) \;[:]\; \phi_1 \mathbin{\dot\vee} \phi_2} \qquad \frac{\Gamma \vdash d \;[:]\; \phi_2}{\Gamma \vdash \vee_r^+(\phi_1, d) \;[:]\; \phi_1 \mathbin{\dot\vee} \phi_2}$$

$$\frac{\Gamma \vdash d \;[:]\; \phi_1 \mathbin{\dot\vee} \phi_2 \quad \Gamma;\phi_1 \vdash e_1 \;[:]\; \chi \quad \Gamma;\phi_2 \vdash e_2 \;[:]\; \chi}{\Gamma \vdash \vee^-(d, e_1, e_2) \;[:]\; \chi}$$

$$\frac{\uparrow\Gamma \vdash d \;[:]\; \phi}{\Gamma \vdash \forall^+(d) \;[:]\; \dot\forall \phi} \qquad \frac{\Gamma \vdash d \;[:]\; \dot\forall \phi}{\Gamma \vdash \forall^-(t, d) \;[:]\; \phi[t]}$$

$$\frac{\Gamma \vdash d \;[:]\; \phi[t]}{\Gamma \vdash \exists^+(\phi, t, d) \;[:]\; \dot\exists \phi} \qquad \frac{\Gamma \vdash d \;[:]\; \dot\exists \chi \quad \uparrow\Gamma;\chi \vdash e \;[:]\; \uparrow\phi}{\Gamma \vdash \exists^-(d, e) \;[:]\; \phi}$$

In contrast to a formalisation with named variables (see [57]), there is a canonical choice of a fresh variable in the setting with De Bruijn indices, as, for example, needed in the rules $\forall^+$ and $\exists^-$. We simply lift all free variables (of Γ in the case of $\forall^+$, and of Γ and ϕ in the case of $\exists^-$), so that the first free variable becomes fresh.

The deduction system above defines the De Bruijn binding mechanism for assumption variables. Binders of assumption variables are $\rightarrow^+$, $\vee^-$ and $\exists^-$. For example, in $\rightarrow^+(\phi, \rightarrow^+(\chi, \wedge^+(h_1, h_0)))$, h_1 refers to the outer $\rightarrow^+$ and h_0 refers to the inner $\rightarrow^+$, as illustrated by the corresponding proof tree.

$$\cfrac{\cfrac{\cfrac{\phi;\chi \vdash h_1 \;[:]\; \phi \quad \phi;\chi \vdash h_0 \;[:]\; \chi}{\phi;\chi \vdash \wedge^+(h_1, h_0) \;[:]\; \phi \mathbin{\dot\wedge} \chi}}{\phi \vdash \rightarrow^+(\chi, \wedge^+(h_1, h_0)) \;[:]\; \chi \mathbin{\dot\rightarrow} \phi \mathbin{\dot\wedge} \chi}}{\vdash \rightarrow^+(\phi, \rightarrow^+(\chi, \wedge^+(h_1, h_0))) \;[:]\; \phi \mathbin{\dot\rightarrow} \chi \mathbin{\dot\rightarrow} \phi \mathbin{\dot\wedge} \chi}$$

Note that $\vee^-$ and $\exists^-$ don't bind assumption variables in their first argument, since in the subtrees corresponding to those arguments no assumption is introduced (travelling bottom-up) into the context. For example, in

$$\rightarrow^+(\phi \mathbin{\dot\vee} \chi, \vee^-(h_0, \vee_r^+(\chi, \underline{h_0}), \vee_l^+(\phi, \underline{h_0})))$$

only the underlined ocurrences of h_0 are bound by the constructor $\vee^-$; the other one (referring to $\phi \mathbin{\dot\vee} \chi$) is bound by the constructor $\rightarrow^+$. The corresponding proof tree is:

$$\cfrac{\cfrac{\phi \mathbin{\dot\vee} \chi \vdash h_0 \;[:]\; \phi \mathbin{\dot\vee} \chi \quad \mathcal{T}_1 \quad \mathcal{T}_2}{\phi \mathbin{\dot\vee} \chi \vdash \vee^-(h_0, \vee_r^+(\chi, h_0), \vee_l^+(\phi, h_0)) \;[:]\; \chi \mathbin{\dot\vee} \phi}}{\vdash \rightarrow^+(\phi \mathbin{\dot\vee} \chi, \vee^-(h_0, \vee_r^+(\chi, h_0), \vee_l^+(\phi, h_0))) \;[:]\; \phi \mathbin{\dot\vee} \chi \mathbin{\dot\rightarrow} \chi \mathbin{\dot\vee} \phi}$$

where $\mathcal{T}_1$ denotes

$$\dfrac{\phi \mathbin{\dot\vee} \chi; \phi \vdash h_0\ [:]\ \phi}{\phi \mathbin{\dot\vee} \chi; \phi \vdash \vee_r^+(\chi, h_0)\ [:]\ \chi \mathbin{\dot\vee} \phi}$$

and $\mathcal{T}_2$ is the analogous tree of $\phi \mathbin{\dot\vee} \chi; \chi \vdash \vee_l^+(\phi, h_0)\ [:]\ \chi \mathbin{\dot\vee} \phi$.

Some constructors also bind *term* variables. The constructor $\forall^+$ binds the first free variable in its argument; $\exists^-$ binds the first free variable in its second argument. These variables are called the *eigenvariables* of $\forall^+$ and $\exists^-$. The constructor $\exists^+$ binds the first free term variable in its first argument. We give a final example:

$$\dfrac{\dfrac{\dfrac{\dot\exists\uparrow_1\phi \mathrel{\dot\to} \uparrow\chi; \phi \vdash h_1\ [:]\ \dot\exists\uparrow_1\phi \mathrel{\dot\to} \uparrow\chi \qquad \dfrac{\dot\exists\uparrow_1\phi \mathrel{\dot\to} \uparrow\chi; \phi \vdash h_0\ [:]\ \phi}{\dot\exists\uparrow_1\phi \mathrel{\dot\to} \uparrow\chi; \phi \vdash \exists^+(\uparrow_1\phi, v_0, h_0)\ [:]\ \dot\exists\uparrow_1\phi}}{\dot\exists\uparrow_1\phi \mathrel{\dot\to} \uparrow\chi; \phi \vdash \to^-(h_1, \exists^+(\uparrow_1\phi, v_0, h_0))\ [:]\ \uparrow\chi}}{\dot\exists\uparrow_1\phi \mathrel{\dot\to} \uparrow\chi \vdash \to^+(\phi, \to^-(h_1, \exists^+(\uparrow_1\phi, v_0, h_0)))\ [:]\ \phi \mathrel{\dot\to} \uparrow\chi}}{\dot\exists\,\phi \mathrel{\dot\to} \chi \vdash \forall^+(\to^+(\phi, \to^-(h_1, \exists^+(\uparrow_1\phi, v_0, h_0))))\ [:]\ \dot\forall\,(\phi \mathrel{\dot\to} \uparrow\chi)}$$

The application of the $\exists^+$-rule is correct as $(\uparrow_1\phi)[v_0] = \phi$ by the following lemma. The application of $\forall^+$ is correct because $\uparrow(\dot\exists\,\phi \mathrel{\dot\to} \chi) = \dot\exists\uparrow_1\phi \mathrel{\dot\to} \uparrow\chi$, by definition of lifting. Note that, on the named level, for the formula $(\exists x.\phi \to \chi) \to \forall x.(\phi \to \chi)$ to be a tautology, the condition $x \notin \mathsf{FV}(\chi)$ is required to avoid capture by $\forall x$ in χ. This is expressed by $\uparrow\chi$ and can be compared to $\curlywedge x.\chi$ (see Chapter 3).

Lemma 2.6.1 *For $n : \mathbb{N}$, $t : \tau$ as well as for $t : o$, $(\uparrow_{n+1}t)[v_0]^n = t$.*

Note that, because intuitionistic predicate logic has the structural rules of weakening, exchange and contraction, the formulation of natural deduction above is logically equivalent to one that mentions (possibly) different contexts in rules with more than one premiss. Given the structural rules (shown to be derivable in the meta-theory in Section 2.7), for example, the following formulation of the rule for $\to^-$ is admissable, as can be shown by applying the weakening lemma (Lemma 2.7.2) $|\Gamma'|$ times. The assumption variables in d have to be lifted so that they still refer to the same assumptions in Γ as they originally did. Proof term e can be left unchanged, because there are no other assumption variables in e than those referring to Γ'.

$$\dfrac{\Gamma \vdash d\ [:]\ \phi \mathrel{\dot\to} \chi \qquad \Gamma' \vdash e\ [:]\ \phi}{\Gamma, \Gamma' \vdash \to^-(\uparrow_0^{|\Gamma'|} d, e)\ [:]\ \chi}$$

In Section 2.11 we show soundness of the deduction relation given in Definition 2.6.1, with respect to an interpetation function $[\![\_]\!]$ mapping object-level formulas to Coq's native logic.

2.7 Admissible Rules

The following rules are *admissible*, that is, derivable in the meta-theory. In order to prove by induction, the statements are loaded appropriately (quantification over Γ, Δ, and so forth).

Lemma 2.7.1 (Lifting of judgement)

$$\frac{\Gamma \vdash d \,[:]\, \phi}{\uparrow_n \Gamma \vdash \uparrow^{\mathsf{v}}_n d \,[:]\, \uparrow_n \phi}$$

Proof. Induction on the proposition $\Gamma \vdash d \,[:]\, \phi$. The proofs of cases $\forall^+$ and $\exists^-$ require Lemma 2.5.1; cases $\forall^-$ and $\exists^+$ require Lemma 2.5.4.

Lemma 2.7.2 (Weakening)

$$\frac{\Gamma; \Delta \vdash d \,[:]\, \phi}{\Gamma; \chi; \Delta \vdash \uparrow^{\mathsf{h}}_{|\Delta|} d \,[:]\, \phi}$$

Proof. By induction on d and inverting the judgement.

Lemma 2.7.3 (Substitution of variables v_i in derivation terms)

$$\frac{\uparrow_n \Gamma; \Delta \vdash d \,[:]\, \phi}{\Gamma; \Delta[t]^n \vdash d[t]^n_{\mathsf{v}} \,[:]\, \phi[t]^n}$$

Proof. By induction on d and inversion. Case h_i is proved by induction over i and Lemma 2.5.2. Cases $\forall^+$ and $\exists^-$ require lemmas 2.5.3 and 2.5.1. Cases $\forall^-$ and $\exists^-$ require Lemma 2.5.5.

Lemma 2.7.4 (Substitution of variables h_i in derivation terms)

$$\frac{\Gamma \vdash d \,[:]\, \phi \quad \Gamma; \phi; \Delta \vdash e \,[:]\, \chi}{\Gamma; \Delta \vdash e[d]^{|\Delta|}_{\mathsf{h}} \,[:]\, \chi}$$

Proof. By induction on e and inversion. Case h_i is proved by induction over i and Lemma 2.7.2.

Exchange, contraction

The structural rules *exchange* and *contraction* are admissible, too.[11] First we need the functions exch and contr. The former swaps the indices n and $n+1$, while the latter decrements all indices greater than n, where n intends to be the reference depth of assumption variables ($n = |\Delta|$ in Lemmas 2.7.5 and 2.7.6).

$$\mathsf{exch}(n,i) = \begin{cases} h_{n+1} & \text{if } i = n \\ h_n & \text{if } i = n+1 \\ h_i & \text{otherwise} \end{cases} \qquad \mathsf{contr}(n,i) = \begin{cases} h_{i-1} & \text{if } i > n \\ h_i & \text{otherwise} \end{cases}$$

Again, the side conditions in the definitions above are avoided in the formalisation.

[11] These lemmas are not needed in the proof of Subject Reduction (Thm. 2.16.2).

Lemma 2.7.5 (Exchange)

$$\frac{\Gamma; \chi; \phi; \Delta \vdash d \; [:] \; \psi}{\Gamma; \phi; \chi; \Delta \vdash \mathsf{map}^{\mathsf{h}}_{\pi}(\mathsf{exch}, |\Delta|, d) \; [:] \; \psi}$$

Lemma 2.7.6 (Contraction)

$$\frac{\Gamma; \phi; \phi; \Delta \vdash d \; [:] \; \chi}{\Gamma; \phi; \Delta \vdash \mathsf{map}^{\mathsf{h}}_{\pi}(\mathsf{contr}, |\Delta|, d) \; [:] \; \chi}$$

2.8 Translation to Coq's Native Logic

We define the translation of object level statements (i.e., the objects defined in Definition 2.2.3) to meta-level statements (i.e., in the language of the framework itself). This translation will be referred to as *interpretation* and depends on a set A, the domain of discourse and parameters $\mathcal{V}$, $\mathcal{F}$, $\mathcal{R}$ for interpreting variables, function symbols and relation symbols respectively. As will be explained in Subsection 2.9.1, using $\mathbb{N}$ as an index set for variables, requires the domain to be non-empty; choose a_0 as the default value in A.

We introduce the operations of *shifting*, notation $\Uparrow_n \mathcal{V}$ and *inserting* terms $a : A$, notation $\mathcal{V}[a]^n$, in variable mappings, that is, λ-terms of type $\mathbb{N} \rightarrow A$.

Definition 2.8.1 *Given $\mathcal{V} : \mathbb{N} \rightarrow A$, $n : \mathbb{N}$, we define $\Uparrow_n \mathcal{V}$ as follows.*

$$\Uparrow_n \mathcal{V} = \lambda p{:}\mathbb{N}.\, \mathcal{V}(p + n)$$

Definition 2.8.2 *Given $\mathcal{V} : \mathbb{N} \rightarrow A$, $n : \mathbb{N}$ and $a : A$, $\mathcal{V}[a]^n$ is defined as follows.*

$$\begin{array}{rcl} \mathcal{V}[a]^0(0) & = & a \\ \mathcal{V}[a]^0(m+1) & = & \mathcal{V}(m) \\ \mathcal{V}[a]^{n+1}(0) & = & \mathcal{V}(0) \\ \mathcal{V}[a]^{n+1}(m+1) & = & (\Uparrow_1 \mathcal{V})[a]^n(m) \end{array}$$

We write $\mathcal{V}[x]$ for $\mathcal{V}[x]^0$.

Term evaluation is defined as follows.

Definition 2.8.3 *Assume an arbitrary domain of discourse $A : *^s$ and a function $\mathcal{V} : \mathbb{N} \rightarrow A$ to interpret (free) variables. Declare a parameter $\mathcal{F}$, a family of functions indexed over $I_{l_{\mathsf{fun}}}$, used to interpret function symbols.*

$$\mathcal{F} : \Pi i{:}I_{l_{\mathsf{fun}}}.\, A^{l_{\mathsf{fun}}(i)} \rightarrow A$$

We write $\mathcal{F}_i$ for $(\mathcal{F}\ i)$. Given such a family, we define the evaluation function for terms of type τ.

$$\begin{array}{rcl} [\![v_n]\!]^{\mathcal{V}} & = & \mathcal{V}(n) \\ [\![f_i(t_1, \ldots, t_k)]\!]^{\mathcal{V}} & = & \mathcal{F}_i([\![t_1]\!]^{\mathcal{V}}, \ldots, [\![t_k]\!]^{\mathcal{V}}) \end{array}$$

Next, we define the canonical interpretation of objects of type o.

Definition 2.8.4 *Again, let $A : *^s$ and $\mathcal{V} : \mathbb{N} \to A$. Assume a family of relations indexed over $I_{l_{\mathsf{rel}}}$.*

$$\mathcal{R} : \Pi j\!:\!I_{l_{\mathsf{rel}}}.\, A^{l_{\mathsf{rel}}(j)} \to *^p$$

We write $\mathcal{R}_i$ for $(\mathcal{R}\ i)$.

$$\begin{array}{rcl}
[\![\dot{\top}]\!]^{\mathcal{V}} & = & \top \\
[\![\dot{\bot}]\!]^{\mathcal{V}} & = & \bot \\
[\![R_j(t_1, \ldots, t_m)]\!]^{\mathcal{V}} & = & \mathcal{R}_j([\![t_1]\!]^{\mathcal{V}}, \ldots, [\![t_m]\!]^{\mathcal{V}}) \\
[\![\phi \mathbin{\dot{\wedge}} \chi]\!]^{\mathcal{V}} & = & [\![\phi]\!]^{\mathcal{V}} \wedge [\![\chi]\!]^{\mathcal{V}} \\
[\![\phi \mathbin{\dot{\vee}} \chi]\!]^{\mathcal{V}} & = & [\![\phi]\!]^{\mathcal{V}} \vee [\![\chi]\!]^{\mathcal{V}} \\
[\![\phi \mathbin{\dot{\to}} \chi]\!]^{\mathcal{V}} & = & [\![\phi]\!]^{\mathcal{V}} \to [\![\chi]\!]^{\mathcal{V}} \\
[\![\dot{\forall}\, \phi]\!]^{\mathcal{V}} & = & \Pi x\!:\!A.\, [\![\phi]\!]^{\mathcal{V}[x]} \\
[\![\dot{\exists}\, \phi]\!]^{\mathcal{V}} & = & \exists x\!:\!A.\, [\![\phi]\!]^{\mathcal{V}[x]}
\end{array}$$

Initially (for closed formulas) we set $\mathcal{V}_0 = \lambda n\!:\!\mathbb{N}.\, a_0$, with a_0 the chosen default value in A, and define $[\![\phi]\!] = [\![\phi]\!]^{\mathcal{V}_0}$.

We use $\top, \bot, \wedge, \vee, \exists$ for Coq's predefined logical connectives. Note that '$\to$' (and 'Π') is used for both (dependent) function space as well as for logical implication (quantification); this overloading witnesses the Curry–Howard–De Bruijn isomorphism.

We don't have to worry about name conflicts when inserting a new $x : A$ to the variable interpretation function $\mathcal{V}$ (quantifier cases). Coq's binding mechanisms are internally based on De Bruijn indices (with a user-friendly tool showing named variables on top of it).

Definition 2.8.5 *The interpretation of a context is the conjunction of its interpreted elements.*

$$[\![\Box]\!]^{\mathcal{V}} = \top \qquad [\![\Gamma; \phi]\!]^{\mathcal{V}} = [\![\Gamma]\!]^{\mathcal{V}} \wedge [\![\phi]\!]^{\mathcal{V}}$$

Remark 2.8.1 *We stress the following analogies between the types of v, f, R, and the types of $\mathcal{V}$, $\mathcal{F}$, $\mathcal{R}$, respectively. First note that:*

$$\begin{array}{rccl}
v_n & \textit{is syntactic sugar for} & & (v\ n) \\
f_i(t_1, \ldots, t_k) & \text{''} & \text{''} & (f\ i\ t_1\ \ldots\ t_k) \\
R_j(t_1, \ldots, t_m) & \text{''} & \text{''} & (R\ j\ t_1\ \ldots\ t_m)
\end{array}$$

(Recall that $k = l_{\mathsf{fun}}(i)$ and $m = l_{\mathsf{rel}}(j)$.)

$$\begin{array}{rccl}
v : \mathbb{N} \to \tau & \textit{analogous to} & & \mathcal{V} : \mathbb{N} \to A \\
f : \Pi i\!:\!I_{l_{\mathsf{fun}}}.\, \tau^k \to \tau & \text{''} & \text{''} & \mathcal{F} : \Pi i\!:\!I_{l_{\mathsf{fun}}}.\, A^k \to A \\
R : \Pi j\!:\!I_{l_{\mathsf{rel}}}.\, \tau^m \to o & \text{''} & \text{''} & \mathcal{R} : \Pi j\!:\!I_{l_{\mathsf{rel}}}.\, A^m \to *^p
\end{array}$$

2.9 Free Variables

2.9.1 Free Variables, Finitely versus Infinitely Many

Note that, differently from type theory where variables have to be declared in the environment, in our representation we have infinitely many variables ($\mathbb{N}$ is the index set of variables). Therefore, we shall need a default value in order to have a total evaluation function (see Definition 2.8.3). Alternatively, we could have chosen to parameterise the sets of terms, formulas, and proof terms over a natural number n indicating the number of free variables an object is allowed to contain (enforced by definition via dependent types). Variables would then be indexed over $\mathbb{N}_n$, defined as follows (think of $\mathbb{N}_n$ as $\{0, \ldots, n-1\}$).

$$\mathbb{N}_0 = \emptyset \qquad \mathbb{N}_{n+1} = \mathbf{1} + \mathbb{N}_n$$

The set τ_n of first-order terms containing n free variables would then be defined as follows; let $m : \mathbb{N}_n$ and $t_1, \ldots, t_k : \tau_n$.

$$\tau_n := v_m \mid f_i(t_1, \ldots, t_k)$$

The constructors $\dot{\forall}$ and $\dot{\exists}$ of o_n then should be typed $o_{n+1} \to o_n$, as they bind the first free variable of their argument. The definition of lifting should be such that, given $t : \tau_n$, the application $\uparrow_m t$ is typed τ_{n+1} (a fresh variable v_m is introduced) and that $m \leq n$ is enforced. Given $k, m : \mathbb{N}$, $t : \tau_{k+m+1}$ and $t' : \tau_k$, $t[t']^m$ should be typed τ_{k+m}. Apparently, such an extra parameter means a considerable complication of matters and we chose to do without it. As a consequence, to be able to define a $\mathcal{V} : \mathbb{N} \to A$ for the evaluation of objects, one needs a default value in A.

2.9.2 Free Variables and Substitution

The De Bruijn representation works elegantly for bound variables, there is no renaming and the structural equality on De Bruijn terms corresponds to the intented identity of terms. As pointed out in [50], however, there is a slight inconvenience in the way free variables are treated. The point is that the order of free variables matters, not their names.

The subtle point about an expression $t[t']^n$ is that *the first n variables are assumed to be bound.* Let $\mathcal{V} : \mathbb{N} \to A$ be such that $\mathcal{V}(0) = y$ and $\mathcal{V}(1) = x$ (i.e., y is introduced later than x), then we can make a substitution that transforms, for example $\mathcal{R}_j(x, y)$ into $\mathcal{R}_j(x, x)$, as illustrated below. Note that, for any $\mathcal{V}'$, if t is interpreted under $\mathcal{V}'$, then $t[t']$ has to be interpreted under $\Uparrow_1 \mathcal{V}'$, because the original occurrences of v_0 in t that pointed to $\mathcal{V}'(0)$ have been removed, and the other variables have been decremented. We have $\Uparrow_1 \mathcal{V}(0) = \mathcal{V}(1) = x$ and

$$[\![R_j(v_1, v_0)]\!]^{\mathcal{V}} = \mathcal{R}_j(x, y)$$

$$[\![R_j(v_1, v_0)[v_0]]\!]^{\Uparrow_1 \mathcal{V}} = [\![R_i(v_0, v_0)]\!]^{\Uparrow_1 \mathcal{V}} = \mathcal{R}_j(x, x)$$

However, we *cannot* make a substitution that transforms $\mathcal{R}_j(x,y)$ into $\mathcal{R}_j(y,y)$. The reason for this is that x corresponds to v_1 and if you want to replace this, it is assumed that v_0 (pointing to y) is bound so that the variables in the substituent are lifted.

The substitution functions are meant for use only in combination with the removal of a binder; $\phi[t]$ is called to instantiate $\dot\forall\phi$ with t or to give t as a witness for $\dot\exists\phi$. Another possible (meta-level) binder is the variable mapping $\mathcal{V}$ as exemplified above. We maintain the term "substitution" *par abus de langage.*

2.10 Thinning and Substitution Lemmas

It is possible to insert free variables to the mapping $\mathcal{V}$ of the interpretation function given in definitions 2.8.3, 2.8.4 and, if the argument is appropriately lifted, keep the same interpretations. This is called *thinning* and can be compared to *weakening* (see Lemma 2.7.2); the latter is about assumption variables, the former about term variables. First we define some auxiliary lemmas.

Lemma 2.10.1 *For all $\mathcal{V},\mathcal{V}' : \mathbb{N} \to A$, $x,y : A$, $n,m : \mathbb{N}$, $t:\tau$ and $\phi : o$, we have:*

$$\begin{array}{rcll}
(\mathcal{V}[x]^n)[y](m) & = & (\mathcal{V}[y])[x]^{n+1}(m) & \textit{(permutation of insertion)}\\
\Uparrow_{n+1}(\mathcal{V}[x])(m) & = & \Uparrow_n\mathcal{V}(m) & \textit{(simplification of insertion)}\\
[\![\uparrow t]\!]^{\mathcal{V}} & = & [\![t]\!]^{\Uparrow_1\mathcal{V}} & \textit{(lift-shift interchange)}
\end{array}$$

Extensional equality of $\mathcal{V}$ and $\mathcal{V}'$, i.e. $\Pi n.\,\mathcal{V}(n) = \mathcal{V}'(n)$, implies $[\![t]\!]^{\mathcal{V}} = [\![t]\!]^{\mathcal{V}'}$ and $[\![\phi]\!]^{\mathcal{V}} \leftrightarrow [\![\phi]\!]^{\mathcal{V}'}$.

Lemma 2.10.2 (Thinning lemma) *Let $\mathcal{V} : \mathbb{N} \to A$, $a : A$ and $n : \mathbb{N}$. (Analogous to Lemma 2.7.2).*

$$\begin{array}{rcl}
[\![t]\!]^{\mathcal{V}} & = & [\![\uparrow_n t]\!]^{\mathcal{V}[a]^n}\\
[\![\phi]\!]^{\mathcal{V}} & \leftrightarrow & [\![\uparrow_n \phi]\!]^{\mathcal{V}[a]^n}
\end{array}$$

Similarly we need $[\![t[t']]\!]^{\mathcal{V}} = [\![t]\!]^{\mathcal{V}[[\![t']\!]^{\mathcal{V}}]}$. We need induction loading, no longer assuming that $[\![t']\!]^{\mathcal{V}}$ is the last added element.

Lemma 2.10.3 (Substitution lemma) *(Analogous to Lemma 2.7.4).*

$$\begin{array}{rcl}
[\![t[t']^n]\!]^{\mathcal{V}} & = & [\![t]\!]^{\mathcal{V}[[\![t']\!]^{\Uparrow_n\mathcal{V}}]^n}\\
[\![\phi[t']^n]\!]^{\mathcal{V}} & \leftrightarrow & [\![\phi]\!]^{\mathcal{V}[[\![t']\!]^{\Uparrow_n\mathcal{V}}]^n}
\end{array}$$

2.11 Soundness with respect to the Native Logic

We show that the deduction relation defined in Definition 2.6.1 is sound with respect to Coq's native logic.

Theorem 2.11.1 (Soundness) *For all contexts Γ, proof terms d, formulas ϕ, and variable mappings $\mathcal{V}$ we have that:*

$$(\Gamma \vdash d\ [:]\ \phi) \to [\![\Gamma]\!]^{\mathcal{V}} \to [\![\phi]\!]^{\mathcal{V}}$$

Proof. First the statement is loaded to $(\Gamma \vdash d\ [:]\ \phi) \to \Pi\mathcal{V}\!:\!\mathbb{N} \to A.\, [\![\Gamma]\!]^{\mathcal{V}} \to [\![\phi]\!]^{\mathcal{V}}$. Its proof proceeds by induction on the proposition $\Gamma \vdash d\ [:]\ \phi$. We sketch the proof for some representative cases.

(Case $\Gamma \vdash \to^{+}(\phi_1, d)\ [:]\ \phi_1 \dot{\to} \phi_2$) Assume $H_\Gamma : [\![\Gamma]\!]^{\mathcal{V}}$. We have the induction hypothesis $IH_d : [\![\Gamma; \phi_1]\!]^{\mathcal{V}} \to [\![\phi_2]\!]^{\mathcal{V}}$. Note that $[\![\Gamma; \phi_1]\!]^{\mathcal{V}} = [\![\Gamma]\!]^{\mathcal{V}} \wedge [\![\phi_1]\!]^{\mathcal{V}}$. It suffices to prove $[\![\phi_1]\!]^{\mathcal{V}} \to [\![\phi_2]\!]^{\mathcal{V}}$. Assume $H_{\phi_1} : [\![\phi_1]\!]^{\mathcal{V}}$, then IH_d applied to the pair $\langle H_\Gamma, H_{\phi_1}\rangle$, is a proof of $[\![\phi_2]\!]^{\mathcal{V}}$.

(Case $\Gamma \vdash \vee^{-}(d, e_1, e_2)\ [:]\ \phi$) Assume $H_\Gamma : [\![\Gamma]\!]^{\mathcal{V}}$. The proof obligation is $[\![\phi]\!]^{\mathcal{V}}$. Three induction hypotheses, corresponding to the three premisses of the $\vee^{-}$-rule are $IH_d : [\![\Gamma]\!]^{\mathcal{V}} \to [\![\chi_1 \dot{\vee} \chi_2]\!]^{\mathcal{V}}$ and $IH_{e_i} : [\![\Gamma; \chi_i]\!]^{\mathcal{V}} \to [\![\phi]\!]^{\mathcal{V}}$ $(i = 1, 2)$. We get $[\![\chi_1]\!]^{\mathcal{V}} \vee [\![\chi_2]\!]^{\mathcal{V}}$ from IH_d and H_Γ.

- Suppose $H_{\chi_1} : [\![\chi_1]\!]^{\mathcal{V}}$, then $(IH_{e_1}\ \langle H_\Gamma, H_{\chi_1}\rangle) : [\![\phi]\!]^{\mathcal{V}}$.
- Suppose $H_{\chi_2} : [\![\chi_2]\!]^{\mathcal{V}}$, then $(IH_{e_2}\ \langle H_\Gamma, H_{\chi_2}\rangle) : [\![\phi]\!]^{\mathcal{V}}$.

(Case $\Gamma \vdash \forall^{+}(d)\ [:]\ \dot{\forall}\,\phi$) Let $H_\Gamma : [\![\Gamma]\!]^{\mathcal{V}}$. The induction hypothesis is $IH_d : \Pi\mathcal{V}\!:\! \mathbb{N} \to A.\, [\![\uparrow\!\Gamma]\!]^{\mathcal{V}} \to [\![\phi]\!]^{\mathcal{V}}$. We have to prove $\Pi x : A.\, [\![\phi]\!]^{\mathcal{V}[x]}$. Assume an arbitrary $x : A$. From Lemma 2.10.2 and H_Γ, it follows that $[\![\uparrow\!\Gamma]\!]^{\mathcal{V}[x]}$. Then, IH_d for $\mathcal{V}[x]$ and the proof of $[\![\uparrow\!\Gamma]\!]^{\mathcal{V}[x]}$, proves $[\![\phi]\!]^{\mathcal{V}[x]}$.

(Case $\Gamma \vdash \exists^{+}(\phi, t, d)\ [:]\ \dot{\exists}\,\phi$) Let $H_\Gamma : [\![\Gamma]\!]^{\mathcal{V}}$. We have $IH_d : [\![\Gamma]\!]^{\mathcal{V}} \to [\![\phi[t]]\!]^{\mathcal{V}}$. The proof obligation is $\exists x\!:\!A.\, [\![\phi]\!]^{\mathcal{V}[x]}$. Give $[\![t]\!]^{\mathcal{V}}$ as witness for this existential statement, so that our goal becomes $[\![\phi]\!]^{\mathcal{V}[[\![t]\!]^{\mathcal{V}}]}$, which, by Lemma 2.10.3, is implied by $[\![\phi[t]]\!]^{\mathcal{V}}$, which in turn follows directly from IH_d and H_Γ.

The following remark explains this chapter's title.

Remark 2.11.1 *As with all lemmas and theorems in this thesis, the proof of Theorem 2.11.1 is a formalised and verified λ-term in Coq:*

$$\mathsf{sound} : \Pi\Gamma, d, \phi.(\Gamma \vdash d\ [:]\ \phi) \to [\![\Gamma]\!] \to [\![\phi]\!]$$

Given H_d of type $\Gamma \vdash d\ [:]\ \phi$ and H_Γ of type $[\![\Gamma]\!]$ for some context Γ, proof term d, and formula ϕ, define:

$$M = (\mathsf{sound}\ \Gamma\ d\ \phi\ H_d\ H_\Gamma)$$

We say that d reflects *the* proof *M of the first-order proposition $[\![\phi]\!]$:*

$$(H_d\!:\!(\Gamma \vdash d\ [:]\ \phi); H_\Gamma\!:\![\![\Gamma]\!]) \vdash_{\mathsf{cic}} M : [\![\phi]\!]$$

where we use $\vdash_{\mathsf{cic}}$ to denote derivability in the calculus of inductive constructions.

For *correct* derivation terms d, the λ-term H_d of type $\Gamma \vdash d\ [:]\ \phi$ can be generated from d, as will be shown in the next two subsections.

2.12 Type Checking Function

Given a context Γ and a proof term d, it is possible to determine whether d reflects a correct proof and, if it does, to synthesise the type of d. First we define so-called options.

Definition 2.12.1 *The set* opt *of* options *is defined inductively as follows. Let* $\phi : o$*.*

$$\mathsf{opt} := \mathsf{val}(\phi) \mid \mathsf{err}$$

Definition 2.12.2 *We define the* type checking function $\mathsf{chk}(\Gamma, d) : \mathsf{opt}$ *by recursion on* d*.*

$$\begin{array}{rcll}
\mathsf{chk}(\Gamma, \top^+) & = & \mathsf{val}(\dot{\top}) & \\
\mathsf{chk}(\Gamma, h_i) & = & \mathsf{val}(\Gamma(i)) & \textit{if } i < |\Gamma| \\
\mathsf{chk}(\Gamma, \bot^-(d,\phi)) & = & \mathsf{val}(\phi) & \textit{if } \mathsf{chk}(\Gamma, d) = \mathsf{val}(\dot{\bot}) \\
\mathsf{chk}(\Gamma, \rightarrow^+(\phi, d)) & = & \mathsf{val}(\phi \dot{\rightarrow} \chi) & \textit{if } \mathsf{chk}([\Gamma;\phi], d) = \mathsf{val}(\chi) \\
\mathsf{chk}(\Gamma, \rightarrow^-(d, e)) & = & \mathsf{val}(\chi) & \textit{if } \left\{\begin{array}{l} \mathsf{chk}(\Gamma, d) = \mathsf{val}(\phi \dot{\rightarrow} \chi) \\ \mathsf{chk}(\Gamma, e) = \mathsf{val}(\phi') \\ \phi = \phi' \end{array}\right. \\
\mathsf{chk}(\Gamma, \vee^-(d, e_1, e_2)) & = & \mathsf{val}(\phi) & \textit{if } \left\{\begin{array}{l} \mathsf{chk}(\Gamma, d) = \mathsf{val}(\psi_1 \dot{\vee} \psi_2) \\ \mathsf{chk}([\Gamma;\psi_1], e_1) = \mathsf{val}(\phi) \\ \mathsf{chk}([\Gamma;\psi_2], e_2) = \mathsf{val}(\phi') \\ \phi' = \phi \end{array}\right. \\
\mathsf{chk}(\Gamma, \forall^+(d)) & = & \mathsf{val}(\dot{\forall}\,\phi) & \textit{if } \mathsf{chk}(\uparrow\Gamma, d) = \mathsf{val}(\phi) \\
\mathsf{chk}(\Gamma, \forall^-(t, d)) & = & \mathsf{val}(\phi[t]) & \textit{if } \mathsf{chk}(\Gamma, d) = \mathsf{val}(\dot{\forall}\,\phi) \\
\mathsf{chk}(\Gamma, \exists^+(\phi, t, d)) & = & \mathsf{val}(\dot{\exists}\,\phi) & \textit{if } \left\{\begin{array}{l} \mathsf{chk}(\Gamma, d) = \mathsf{val}(\phi') \\ \phi' = \phi[t] \end{array}\right. \\
\mathsf{chk}(\Gamma, \exists^-(d, e)) & = & \mathsf{val}(\downarrow\phi) & \textit{if } \left\{\begin{array}{l} \mathsf{chk}(\Gamma, d) = \mathsf{val}(\dot{\exists}\,\chi) \\ \mathsf{chk}([\uparrow\Gamma;\chi], e) = \mathsf{val}(\phi) \\ v_0 \notin \mathsf{FV}(\phi) \end{array}\right.
\end{array}$$

For any recursive call on a subterm, it is checked whether it gives a value or an error. Thus, unlike other programming languages, errors have to be propagated recursively. The proviso's are defined by case analysis on the recursive calls on substructures and by using a Boolean equality relation on formulas. If these conditions are not satisfied, err is returned. The canonical cases for the constructors $\wedge^+$, $\wedge_l^-$, $\wedge_r^-$, $\vee_l^+$, $\vee_r^+$ are left out.

2.13 Correctness of Type Checking Function

Type checking is sound and complete with respect to the deduction system of Definition 2.6.1.

Theorem 2.13.1 (Correctness of chk) *For all proof terms d, contexts Γ and formulas ϕ, we have that:*

$$\mathsf{chk}(\Gamma, d) = \mathsf{val}(\phi) \leftrightarrow \Gamma \vdash d\ [:]\ \phi$$

Proof. ($\rightarrow$) By induction on d.

(Case $\mathsf{chk}(\Gamma, \exists^-(d,e)) = \mathsf{val}(\downarrow\phi)$) From this it follows that $\mathsf{chk}(\Gamma, d) = \mathsf{val}(\dot{\exists}\,\chi)$ and $\mathsf{chk}([\uparrow\Gamma;\chi], e) = \mathsf{val}(\phi)$. By the induction hypotheses, we obtain $\Gamma \vdash d\ [:]\ \dot{\exists}\,\chi$ and $\uparrow\Gamma;\chi \vdash e\ [:]\ \phi$. Because e is a correct term, $v_0 \notin \mathsf{FV}(\phi)$; by Lemma 2.4.2 we get $\phi = \uparrow\downarrow\phi$. The proof obligation, then, is fulfilled by application of the inference rule for $\exists^-$.

$$\frac{\Gamma \vdash d\ [:]\ \dot{\exists}\,\chi \quad \uparrow\Gamma;\chi \vdash e\ [:]\ \uparrow\downarrow\phi}{\Gamma \vdash \exists^-(d,e)\ [:]\ \downarrow\phi}$$

($\leftarrow$) By induction on $\Gamma \vdash d\ [:]\ \phi$.

(Case $\Gamma \vdash \exists^-(d,e)\ [:]\ \phi$) We have $\Gamma \vdash d\ [:]\ \dot{\exists}\,\chi$ and $\uparrow\Gamma;\chi \vdash e\ [:]\ \uparrow\phi$. By the induction hypotheses, we obtain $\mathsf{chk}(\Gamma, d) = \mathsf{val}(\dot{\exists}\,\chi)$ and $\mathsf{chk}([\uparrow\Gamma;\chi], e) = \mathsf{val}(\uparrow\phi)$. We have $v_0 \notin \mathsf{FV}(\uparrow\phi)$ (Lemma 2.4.3) and so $\mathsf{chk}(\Gamma, \exists^-(d,e)) = \mathsf{val}(\downarrow\uparrow\phi)$; finally, $\downarrow\uparrow\phi = \phi$ by Lemma 2.4.1.

2.14 Unique Types

Proof terms have unique types.

Corollary 2.14.1 (Uniqueness of Types) *For all proof terms d, contexts Γ and formulas ϕ and χ, we have that:*

$$(\Gamma \vdash d\ [:]\ \phi) \rightarrow (\Gamma \vdash d\ [:]\ \chi) \rightarrow \phi = \chi$$

Proof. Direct from double application of Theorem 2.13.1.

2.15 Proof Reduction

To illustrate how the defined machinery can be used to manipulate proof objects, we define Prawitz's proof reduction rules [61].[12] The goal is to remove detours, as in the following tree.

$$\frac{\dfrac{\Gamma;\phi \vdash d\ [:]\ \chi}{\Gamma \vdash \rightarrow^+(\phi,d)\ [:]\ \phi \dot{\rightarrow} \chi} \quad \Gamma \vdash e\ [:]\ \phi}{\Gamma \vdash \rightarrow^-(\rightarrow^+(\phi,d),e)\ [:]\ \chi}$$

Instead of first assuming ϕ to build a proof d of χ, introduce the implication $\phi \dot{\rightarrow} \chi$, and then eliminate it immediately by plugging in derivation e, we can more directly replace the assumption ϕ in d (represented by the first free assumption variable) by e.

$$\frac{}{\Gamma \vdash d[e]_{\mathsf{h}}\ [:]\ \chi}$$

[12]We actually follow [59], pages 85–88.

The removal of such a direct detour is called a *proper reduction*. There are seven such rewrite rules, where on the left-hand side an introduction of a certain connective is immediately followed by an elimination of that connective. Sometimes, proper redexes are *hidden* by intermediate $\vee^-$ and/or $\exists^-$ rules. Such hidden detours are made direct by a sequence of so-called *permutative conversions*. These conversions pull out the $\vee^-$ and $\exists^-$ rules. After the following definition, we give an example of such a permutative conversion. The proof of Theorem 2.16.1 demonstrates why the various lifting operations are necessary to keep correct proofs.

Definition 2.15.1 Immediate proof reduction, *$d \mapsto e$, is defined by the following rewrite rules. The left-hand sides are called immediate (proper, permutative) redexes and the right-hand sides immediate (proper, permutative) reducts.*
Proper reductions.

$$\begin{array}{rcll}
\to^-(\to^+(\phi, d), e) & \mapsto & d[e]_{\mathsf{h}} & (\mathrm{PR}{\to}) \\
\wedge_l^-(\wedge^+(d_1, d_2)) & \mapsto & d_1 & (\mathrm{PR}{\wedge_1}) \\
\wedge_r^-(\wedge^+(d_1, d_2)) & \mapsto & d_2 & (\mathrm{PR}{\wedge_2}) \\
\vee^-(\vee_l^+(\phi, d), e_1, e_2) & \mapsto & e_1[d]_{\mathsf{h}} & (\mathrm{PR}{\vee_1}) \\
\vee^-(\vee_r^+(\phi, d), e_1, e_2) & \mapsto & e_2[d]_{\mathsf{h}} & (\mathrm{PR}{\vee_2}) \\
\forall^-(t, \forall^+(d)) & \mapsto & d[t]_{\mathsf{v}} & (\mathrm{PR}\forall) \\
\exists^-(\exists^+(\phi, t, d), e) & \mapsto & (e[t]_{\mathsf{v}})[d]_{\mathsf{h}} & (\mathrm{PR}\exists)
\end{array}$$

Permutative conversions.

$$\begin{array}{rcll}
\bot^-(\vee^-(d, e_1, e_2), \phi) & \mapsto & \vee^-(d, \bot^-(e_1, \phi), \bot^-(e_2, \phi)) & (\mathrm{PC}{\vee}{\bot}) \\
\to^-(\vee^-(d, e_1, e_2), g) & \mapsto & \vee^-(d, \to^-(e_1, \uparrow^{\mathsf{h}} g), \to^-(e_2, \uparrow^{\mathsf{h}} g)) & (\mathrm{PC}{\vee}{\to}) \\
\wedge_l^-(\vee^-(d, e_1, e_2)) & \mapsto & \vee^-(d, \wedge_l^-(e_1), \wedge_l^-(e_2)) & (\mathrm{PC}{\vee}{\wedge_1}) \\
\wedge_r^-(\vee^-(d, e_1, e_2)) & \mapsto & \vee^-(d, \wedge_r^-(e_1), \wedge_r^-(e_2)) & (\mathrm{PC}{\vee}{\wedge_2}) \\
\vee^-(\vee^-(d, e_1, e_2), g, h) & \mapsto & \vee^-(d, \vee^-(e_1, \uparrow_1^{\mathsf{h}} g, \uparrow_1^{\mathsf{h}} h), & \\
 & & \quad \vee^-(e_2, \uparrow_1^{\mathsf{h}} g, \uparrow_1^{\mathsf{h}} h)) & (\mathrm{PC}{\vee}{\vee}) \\
\forall^-(t, \vee^-(d, e_1, e_2)) & \mapsto & \vee^-(d, \forall^-(t, e_1), \forall^-(t, e_2)) & (\mathrm{PC}{\vee}\forall) \\
\exists^-(\vee^-(d, e_1, e_2), g) & \mapsto & \vee^-(d, \exists^-(e_1, \uparrow_1^{\mathsf{h}} g), \exists^-(e_2, \uparrow_1^{\mathsf{h}} g)) & (\mathrm{PC}{\vee}\exists) \\
\bot^-(\exists^-(d, e), \phi) & \mapsto & \exists^-(d, \bot^-(e, \uparrow \phi)) & (\mathrm{PC}\exists{\bot}) \\
\to^-(\exists^-(d, e), f) & \mapsto & \exists^-(d, \to^-(e, \uparrow^{\mathsf{h}}(\uparrow^{\mathsf{v}} f))) & (\mathrm{PC}\exists{\to}) \\
\wedge_l^-(\exists^-(d, e)) & \mapsto & \exists^-(d, \wedge_l^-(e)) & (\mathrm{PC}\exists{\wedge_1}) \\
\wedge_r^-(\exists^-(d, e)) & \mapsto & \exists^-(d, \wedge_r^-(e)) & (\mathrm{PC}\exists{\wedge_2}) \\
\vee^-(\exists^-(d, e), f, g) & \mapsto & \exists^-(d, \vee^-(e, \uparrow_1^{\mathsf{h}}(\uparrow^{\mathsf{v}} f), \uparrow_1^{\mathsf{h}}(\uparrow^{\mathsf{v}} g))) & (\mathrm{PC}\exists{\vee}) \\
\forall^-(t, \exists^-(d, e)) & \mapsto & \exists^-(d, \forall^-(\uparrow t, e)) & (\mathrm{PC}\exists\forall) \\
\exists^-(\exists^-(d, e), f) & \mapsto & \exists^-(d, \exists^-(e, \uparrow_1^{\mathsf{h}}(\uparrow_1^{\mathsf{v}} f))) & (\mathrm{PC}\exists\exists)
\end{array}$$

Definition 2.15.2 *We define $\rhd$ as the closure of $\mapsto$ under the construction rules of π. In other words, $d \rhd d'$ holds if d' can be obtained from d by replacing a subterm of d by an immediate reduct of it.*

As an example, consider the following reduction sequence, consisting of rules PC$\vee\exists$ and PR$\exists$ respectively.

$$\exists^-(\vee^-(d, \exists^+(\phi, t, e_1), e_2), g)$$

$$\begin{array}{ll} \rhd & \vee^-(d, \exists^-(\exists^+(\phi, t, e_1), \uparrow^{\mathsf{h}}_1 g), \exists^-(e_2, \uparrow^{\mathsf{h}}_1 g)) \\ \rhd & \vee^-(d, ((\uparrow^{\mathsf{h}}_1 g)[t]_{\mathsf{v}})[e_1]_{\mathsf{h}}, \exists^-(e_2, \uparrow^{\mathsf{h}}_1 g)) \end{array}$$

Let's depict the corresponding proof trees, starting with the permutative redex.

$$\dfrac{\dfrac{\Gamma \vdash d\ [:]\ \psi_1 \mathbin{\dot\vee} \psi_2 \quad \dfrac{\Gamma; \psi_1 \vdash e_1\ [:]\ \phi[t]}{\Gamma; \psi_1 \vdash \exists^+(\phi, t, e_1)\ [:]\ \dot\exists\, \phi} \quad \Gamma; \psi_2 \vdash e_2\ [:]\ \dot\exists\, \phi}{\Gamma \vdash \vee^-(d, \exists^+(\phi, t, e_1), e_2)\ [:]\ \dot\exists\, \phi} \qquad \uparrow\Gamma; \phi \vdash g\ [:]\ \uparrow\chi}{\Gamma \vdash \exists^-(\vee^-(d, \exists^+(\phi, t, e_1), e_2), g)\ [:]\ \chi}$$

The previously hidden detour is made direct, as shown in the following tree, corresponding to the permutative reduct.

$$\dfrac{\Gamma \vdash d\ [:]\ \psi_1 \mathbin{\dot\vee} \psi_2 \quad \mathcal{T}_1 \quad \mathcal{T}_2}{\Gamma \vdash \vee^-(d, \exists^-(\exists^+(\phi, t, e_1), \uparrow^{\mathsf{h}}_1 g), \exists^-(e_2, \uparrow^{\mathsf{h}}_1 g))\ [:]\ \chi}$$

Where $\mathcal{T}_1$ denotes

$$\dfrac{\Gamma; \psi_1 \vdash \exists^+(\phi, t, e_1)\ [:]\ \dot\exists\, \phi \quad \uparrow(\Gamma; \psi_1); \phi \vdash \uparrow^{\mathsf{h}}_1 g\ [:]\ \uparrow\chi}{\Gamma; \psi_1 \vdash \exists^-(\exists^+(\phi, t, e_1), \uparrow^{\mathsf{h}}_1 g)\ [:]\ \chi}$$

and $\mathcal{T}_2$ denotes

$$\dfrac{\Gamma; \psi_2 \vdash e_2\ [:]\ \dot\exists\, \phi \quad \uparrow(\Gamma; \psi_2); \phi \vdash \uparrow^{\mathsf{h}}_1 g\ [:]\ \uparrow\chi}{\Gamma; \psi_2 \vdash \exists^-(e_2, \uparrow^{\mathsf{h}}_1 g)\ [:]\ \chi}$$

Now $\mathcal{T}_1$ contains a direct detour, which reduces to $\mathcal{T}_1'$:

$$\dfrac{}{\Gamma; \psi_1 \vdash ((\uparrow^{\mathsf{h}}_1 g)[t]_{\mathsf{v}})[e_1]_{\mathsf{h}}\ [:]\ \chi}$$

The proof tree corresponding to the final term in the reduction sequence then reads:

$$\dfrac{\Gamma \vdash d\ [:]\ \psi_1 \mathbin{\dot\vee} \psi_2 \quad \mathcal{T}_1' \quad \mathcal{T}_2}{\Gamma \vdash \vee^-(d, ((\uparrow^{\mathsf{h}}_1 g)[t]_{\mathsf{v}})[e_1]_{\mathsf{h}}, \exists^-(e_2, \uparrow^{\mathsf{h}}_1 g))\ [:]\ \chi}$$

2.16 Subject Reduction

Theorem 2.16.1 (Subject Reduction ($\mapsto$))

$$(d \mapsto e) \to (\Gamma \vdash d\ [:]\ \phi) \to (\Gamma \vdash e\ [:]\ \phi)$$

Proof. By induction on the proposition $d \mapsto e$. The so obtained instances of $\Gamma \vdash d\ [:]\ \phi$ are inverted twice. We show some representative cases.

(PR→) The following tree is built bottom-up with the use of inversion, starting at the given judgement $\Gamma \vdash \to^{-}(\to^{+}(\phi, d), e)$ [:] χ in the root. Inverting the root gives $\Gamma \vdash \to^{+}(\phi, d)$ [:] $\phi \dot{\to} \chi$ and $\Gamma \vdash e$ [:] ϕ. Inverting the former judgement gives $\Gamma; \phi \vdash d$ [:] χ.

$$\dfrac{\dfrac{\Gamma; \phi \vdash d \text{ [:] } \chi}{\Gamma \vdash \to^{+}(\phi, d) \text{ [:] } \phi \dot{\to} \chi} \quad \Gamma \vdash e \text{ [:] } \phi}{\Gamma \vdash \to^{-}(\to^{+}(\phi, d), e) \text{ [:] } \chi}$$

We have to prove: $\Gamma \vdash d[e]_{\mathsf{h}}$ [:] χ, which follows from Lemma 2.7.4 by substituting the empty context for Δ:

$$\dfrac{\Gamma \vdash e \text{ [:] } \phi \quad \Gamma; \phi \vdash d \text{ [:] } \chi}{\Gamma \vdash d[e]_{\mathsf{h}} \text{ [:] } \chi}$$

(PR∀) Assume $\Gamma \vdash \forall^{-}(t, \forall^{+}(d))$ [:] $\phi[t]$. Using inversion, we build the following tree.

$$\dfrac{\dfrac{\uparrow\Gamma \vdash d \text{ [:] } \phi}{\Gamma \vdash \forall^{+}(d) \text{ [:] } \dot{\forall}\phi}}{\Gamma \vdash \forall^{-}(t, \forall^{+}(d)) \text{ [:] } \phi[t]}$$

We have to prove: $\Gamma \vdash d[t]_{\mathsf{v}}$ [:] $\phi[t]$, which follows from Lemma 2.7.3 and $\uparrow\Gamma \vdash d$ [:] ϕ (take Δ empty and $n = 0$).

(PC∨∨) Assume $\Gamma \vdash \vee^{-}(\vee^{-}(d, e_1, e_2), g, h)$ [:] ϕ. The proof obligation is:

$$\Gamma \vdash \vee^{-}(d, \vee^{-}(e_1, \uparrow^{\mathsf{h}}_{1} g, \uparrow^{\mathsf{h}}_{1} h), \vee^{-}(e_2, \uparrow^{\mathsf{h}}_{1} g, \uparrow^{\mathsf{h}}_{1} h)) \text{ [:] } \phi$$

We use the following abbreviations.

$$\begin{array}{lcl} \mathcal{J}_d & \equiv & \Gamma \vdash d \text{ [:] } \psi_1 \dot{\vee} \psi_2 \\ \mathcal{J}_g & \equiv & \Gamma; \chi_1 \vdash g \text{ [:] } \phi \\ \mathcal{J}_h & \equiv & \Gamma; \chi_2 \vdash h \text{ [:] } \phi \end{array} \qquad \begin{array}{lcl} \mathcal{J}_{e_1} & \equiv & \Gamma; \psi_1 \vdash e_1 \text{ [:] } \chi_1 \dot{\vee} \chi_2 \\ \mathcal{J}_{e_2} & \equiv & \Gamma; \psi_2 \vdash e_2 \text{ [:] } \chi_1 \dot{\vee} \chi_2 \end{array}$$

After inversion, we come to the following tree.

$$\dfrac{\dfrac{\mathcal{J}_d \quad \mathcal{J}_{e_1} \quad \mathcal{J}_{e_2}}{\Gamma \vdash \vee^{-}(d, e_1, e_2) \text{ [:] } \chi_1 \dot{\vee} \chi_2} \quad \mathcal{J}_g \quad \mathcal{J}_h}{\Gamma \vdash \vee^{-}(\vee^{-}(d, e_1, e_2), g, h) \text{ [:] } \phi}$$

The following tree demonstrates how the goal is deduced.

$$\dfrac{\mathcal{J}_d \quad \mathcal{T}_1 \quad \mathcal{T}_2}{\Gamma \vdash \vee^{-}(d, \vee^{-}(e_1, \uparrow^{\mathsf{h}}_{1} g, \uparrow^{\mathsf{h}}_{1} h), \vee^{-}(e_2, \uparrow^{\mathsf{h}}_{1} g, \uparrow^{\mathsf{h}}_{1} h)) \text{ [:] } \phi}$$

where $\mathcal{T}_1$ denotes the subtree:

$$\dfrac{\mathcal{J}_{e_1} \quad \Gamma; \psi_1; \chi_1 \vdash \uparrow^{\mathsf{h}}_{1} g \text{ [:] } \phi \quad \Gamma; \psi_1; \chi_2 \vdash \uparrow^{\mathsf{h}}_{1} h \text{ [:] } \phi}{\Gamma; \psi_1 \vdash \vee^{-}(e_1, \uparrow^{\mathsf{h}}_{1} g, \uparrow^{\mathsf{h}}_{1} h) \text{ [:] } \phi}$$

and $\mathcal{T}_2$ the analogous deduction of $\Gamma; \psi_2 \vdash \vee^-(e_2, \uparrow^{\mathsf{h}}_1 g, \uparrow^{\mathsf{h}}_1 h)\ [:]\ \phi$. Now it becomes clear why all free assumption variables except the first have to be lifted in, for example, proof term g: $\uparrow^{\mathsf{h}}_1 g$. In $\mathcal{T}_1$ the extra assumption ψ_1 is added to the context. The leafs $\Gamma; \psi_1; \chi_1 \vdash \uparrow^{\mathsf{h}}_1 g\ [:]\ \phi$ and $\Gamma; \psi_1; \chi_2 \vdash \uparrow^{\mathsf{h}}_1 h\ [:]\ \phi$ in $\mathcal{T}_1$ are implied by the judgements $\mathcal{J}_g$ and $\mathcal{J}_h$ respectively, via the weakening lemma (2.7.2).

(PC$\exists\!\to$) Assume $\Gamma \vdash \to^-(\exists^-(d, e), f)\ [:]\ \chi$.

$$\dfrac{\dfrac{\Gamma \vdash d\ [:]\ \dot{\exists}\,\psi \quad \uparrow\!\Gamma; \psi \vdash e\ [:]\ \uparrow\!(\phi \dot{\to} \chi)}{\Gamma \vdash \exists^-(d, e)\ [:]\ \phi \dot{\to} \chi} \qquad \Gamma \vdash f\ [:]\ \phi}{\Gamma \vdash \to^-(\exists^-(d, e), f)\ [:]\ \chi}$$

The conversion PC$\exists\!\to$ puts derivation f in the scope of the $\exists^-$. In the new situation, in order to obtain a correct deduction, f is lifted such that it no longers contains v_0 and h_0 (now referring to the new assumption ψ). We have to prove $\Gamma \vdash \exists^-(d, \to^-(e, \uparrow^{\mathsf{h}}(\uparrow^{\mathsf{v}} f)))\ [:]\ \chi$.

$$\dfrac{\Gamma \vdash d\ [:]\ \dot{\exists}\,\psi \qquad \dfrac{\uparrow\!\Gamma; \psi \vdash e\ [:]\ \uparrow\!\phi \dot{\to} \uparrow\!\chi \quad \uparrow\!\Gamma; \psi \vdash \uparrow^{\mathsf{h}}(\uparrow^{\mathsf{v}} f)\ [:]\ \uparrow\!\phi}{\uparrow\!\Gamma; \psi \vdash \to^-(e, \uparrow^{\mathsf{h}}(\uparrow^{\mathsf{v}} f))\ [:]\ \uparrow\!\chi}}{\Gamma \vdash \exists^-(d, \to^-(e, \uparrow^{\mathsf{h}}(\uparrow^{\mathsf{v}} f)))\ [:]\ \chi}$$

Note that $\uparrow\!(\phi \dot{\to} \chi) = \uparrow\!\phi \dot{\to} \uparrow\!\chi$. Thus, all we have to show is that $\uparrow\!\Gamma; \psi \vdash \uparrow^{\mathsf{h}}(\uparrow^{\mathsf{v}} f)\ [:]\ \uparrow\!\phi$ follows from $\Gamma \vdash f\ [:]\ \phi$. By Lemma 2.7.1, we have that $\uparrow\!\Gamma \vdash \uparrow^{\mathsf{v}} f\ [:]\ \uparrow\!\phi$. Then our goal follows from the weakening lemma (2.7.2).

The following theorem, stating that $\rhd$ preserves types, follows directly from Theorem 2.16.1. The proof proceeds by structural induction on the proposition $d \rhd e$.

Theorem 2.16.2 (Subject Reduction ($\rhd$))

$$(d \rhd e) \to (\Gamma \vdash d\ [:]\ \phi) \to (\Gamma \vdash e\ [:]\ \phi)$$

2.17 Conclusion and Future Research

We described a formalisation of natural deduction for intuitionistic first-order logic in Coq. This formalisation provides an object language amenable to the manipulation of formulas and of proof objects, which is the objective of this study. In the meta-theory we are able to reason about these syntactical objects. The example of a proof reduction relation demonstrates how proof terms can be subject to manipulation and to reasoning. Via the soundness (Theorem 2.11.1) of the deduction system of hypothetical judgements (Definition 2.6.1), we are also able to lift object level proof terms to actual proof terms inhabiting propositions of type $*^p$. Thus we can reflect upon the first-order fragment of $*^p$.

We plan to use the described formalisation for a syntactical proof of conservativity of the Axiom of Choice over first-order intuitionistic logic without equality (see [63] and [30]). Also, proving termination of permutative conversions (along the lines of [42] or [59]) is challenging.

Acknowledgments

The author thanks Marc Bezem, Vincent van Oostrom, Jaco van de Pol and Freek Wiedijk for their critical remarks on draft versions of this chapter and for many fruitful discussions.