Proof reflection in Coq

Dimitri Hendriks [*]

December 10, 2001

Abstract

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 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.

1 Introduction

We represent intuitionistic predicate logic in Coq [Coq99], an interactive proof construction system that implements the calculus of inductive constructions [Wer94], which is a type theory that provides inductive definitions. We adopt a two-level approach 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 representing natural deduction proofs. 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.

The major 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 and the soundness result, it's possible to reason about the first-order fragment of the native logic itself. The complete development is formalized in Coq and can be retrieved from URL: `http://www.phil.uu.nl/~hendriks/coq/prfx`.

The paper is organized as follows. In the remainder of the present section we spend some introductory words on type theory and the system Coq, explain the idea of reflection, and motivate our design choices with respect to variable binding mechanisms and the format of hypothetical judgements. In Section 2 we introduce first-order terms, first-order formulas and derivation terms. In Section 3 the definition of substitution is given along with an operation called

[*]Utrecht University, Department of Philosophy, `Dimitri.Hendriks@phil.uu.nl`. The author thanks Marc Bezem, Vincent van Oostrom, Jaco van de Pol and Freek Wiedijk for their critical remarks and suggestions on draft versions of this report.

'lifting'; this section makes clear how the De Bruijn binding mechanism works. The inference rules (constructors) for hypothetical judgements are presented in Section 4. In Section 5 we give an example of how a certain tautology can be deduced. In Section 6 we define the translation from object level formulas to their meta-level counterparts. Section 7 presents thinning and substitution lemma's about this translation function, necessary for the proof of soundness with respect to Coq's logic, given in Section 8. In Section 9, we specify a function which infers the type of (correct) proof terms. In Section 10 this function is proved correct w.r.t. the outlined inference system. As a corollary, derivation terms have unique types (Section 11). Section 12 is devoted to the operations of lifting and substitution of both term variables as well as assumption variables in derivation terms. Section 13 serves as an example of how the defined machinery can be used to manipulate/transform proof terms; Prawitz' proof reduction rules are defined. Sections 14 through 16 list some basic algebraic properties of the defined De Bruijn operations, inversion lemma's and admissable rules (for the relation $\vdash$) respectively. In Section 17 we present soundness of types for the defined proof reduction, the property known as Subject Reduction. Finally, we conclude and discuss future work.

Type theory and Coq

Type theory offers a powerful formalism for formalizing mathematics and in particular for formalizing meta-theory of deduction systems. Definitions, reasoning and computation are captured in an integrated way. The level of detail is such that the well-formedness of definitions and the correctness of derivations can be verified automatically. In a type-theoretical system, formalized mathematical statements are represented by types, and their proofs are represented by λ-terms. The correspondence between natural deduction proofs and typed λ-terms is referred to as the Curry-Howard-(De Bruijn) isomorphism. The relation between a proof and the statement it verifies, can be viewed as the membership of an object in a set. The problem whether a is a proof of statement A reduces to checking whether the term a has type A. An expression is in effect a program annotated with additional information (types), which is used for verification (type checking).

The logical framework of Coq is well-suited for an investigation of the meta-theory of deduction systems such as natural deduction. Useful are the common proof techniques of structural induction, pattern matching and primitive recursion. The user is allowed to extend the type theory with inductive types. Dually, the reduction rules can be extended in a flexible way. An inductive type provides a principle of structural induction, a λ-term automatically generated by the system. Functions whose domain is an inductive type, can be defined using case analysis over the possible constructors of the object and recursion.

The basic sorts in Coq are $*^p$ and $*^s$. An object M of type $*^p$ denotes a logical proposition and objects of type M denote proofs of M. Objects of type $*^s$ are usual sets such as the set of natural numbers, lists etc. The typing relation is expressed by $t : T$, to be interpreted as 't belongs to set T' when $T : *^s$, and as 't is a proof of proposition T' when $T : *^p$. The primitive type constructor is the product type $\Pi x{:}T.\,U$ and is called dependent if x occurs in U; if not, we write $T \to U$. The product type is used for logical quantification (implication) as well as for function spaces, witnessing the Curry-Howard isomorphism. Scopes of Π's, λ's and other binders extend to the right as far as brackets allow ($\rightarrow$ associates to the right). Furthermore, well-typed application is denoted by $(M\ N)$ and associates to the left.

In Coq, connectives are defined as inductive types, the constructors being the proof formators. For example, conjunction $A \wedge B$ is defined as the inductive type inhabited by pairs $\langle a, b\rangle$, where $a : A$ and $b : B$. The corresponding induction principle is inhabited by $\wedge_{ind}$, a lambda term generated by the system.

$$\wedge_{ind} : \Pi A, B, P{:}{*^p}.\, (A \rightarrow B \rightarrow P) \rightarrow A \wedge B \rightarrow P$$

which can be used to eliminate the $\wedge$. For instance, a proof of $A \wedge B \rightarrow B \wedge A$ can be constructed as follows.

$$(\wedge_{ind}\ A\ B\ (B \wedge A)\ (\lambda a{:}A.\, \lambda b{:}B.\, \langle b, a\rangle))$$

Two-level approach, reflection

The universe $*^p$ includes higher-order propositions; in fact it encompasses full impredicative type theory, which is too large for our purposes. Moreover, Coq supplies only limited computational power on $*^p$; every connective is defined as the inductive set of proofs of propositions with that connective in the head. We need a way to grasp first-order formulas and natural deduction proofs, so that they can be subject to syntactical manipulation. Moreover, we want the ability to reason about such objects, and prove logical properties about them.

A natural choice then, is to define formulas and proof terms as inductive objects, equipped with the powerful computational device of higher-order primitive recursion.

Object-level formulas (type o) are related to the meta-level by means of an interpretation function $[\![\_]\!] : o \rightarrow *^p$. Given a suitable signature, any first-order proposition $\phi : *^p$ will have a formal counterpart $p : o$ such that ϕ is convertible with $[\![p]\!]$, the interpretation of p. Thus, the first-order fragment of $*^p$ can be identified as the collection of interpretations of objects in o.

Proof terms are also defined as syntactical objects in an inductive set. The main judgement is of the form $\Gamma \vdash d\ [:]\ \phi$; it is of type $*^p$. The structure of the proof of $\Gamma \vdash d\ [:]\ \phi$ is similar to the structure of d, as will be pointed out in the sequel. Furthermore, we prove that if $\Gamma \vdash d\ [:]\ \phi$, then $[\![\Gamma]\!] \rightarrow [\![\phi]\!]$, in other words we construct a λ-term *sound* of the following type.

$$\mathit{sound} : (\Gamma \vdash d\ [:]\ \phi) \rightarrow [\![\Gamma]\!] \rightarrow [\![\phi]\!]$$

One could say that an object d reflects the λ-term $(\mathit{sound}\ H_d\ H_\Gamma) : [\![\phi]\!]$, where $H_d : (\Gamma \vdash d\ [:]\ \phi)$ and $H_\Gamma : [\![\Gamma]\!]$.

One of the design choices to be made is whether to use a deep or shallow embedding of the objects we need. When syntax and meaning of a language are described separately, the language is said to be deeply embedded. Sometimes it's more economic to use a shallow embedding, where representation and denotation of objects are identified (in other words: the interpretation function is the identity function). The disadvantage of a shallow embedding is that the syntactic structure cannot be exploited.[1] We choose for a deep embedding of

[1]In [Hen98] and[BHN00] a shallow embedding is used for first-order terms. In combination with the use of higher-order abstract syntax to represent quantifiers, this gave rise to several difficulties. For example, it's not possible to prove syntactical correctness of the described formula transformation in a formal way.

terms, formulas and of derivation terms, giving us full control over the defined constructs.

Variable binding

The are several ways to represent the needed variable binding operators (quantification of first-order terms, binding of assumption variables), of which we mention: *naive naming*, *higher-order abstract syntax* and *De Bruijn indexing*. Much of the discussion on the differences between naive naming and representation via De Bruijn indices is based on [Per96].

In informal practice, the so-called *variable convention* plays a crucial role; expressions that differ only in the names assigned to their bound variables are to be identified; $\forall x.\, \phi(x)$ is said to be *α-equivalent* to $\forall y.\, \phi(y)$. In mathematical contexts bound variables are chosen different from free variables. In the process of substitution this means the (silent) renaming of bound variables.

Formalizing bindings via the naive naming approach—where, like in informal mathematics, names (e.g. natural numbers) are used to encode the link between a binder and the bound variable—is technically hard work. On top of the 'natural' definition of formulas one needs to define explicitly α-equality. As pointed out in [Per96], the (unavoidable) use of side-conditions in the definition of substitution is problematic when it comes to computation. As the unfolding of definitions proceeds, the number of side-conditions increases exponentially. Another difficulty is that there is no canonical choice of a fresh variable, necessary for e.g. satisfying the eigenvariable condition in the inference rules $\forall^+$ and $\exists^-$. Moreover, for many applications one needs a way to distinguish free and bound variables.

As for the representation via higher-order abstract syntax, the advantage is that several binding operators are handled by λ-abstraction. Identification of α-convertible formulas now comes for free. Substitution on the object level is supported by β-reduction in the meta-language. One problem of this representation[2] is that it generates a class of terms that contains *too much*. We encountered this problem in [BHN00], where we had constructors $\dot{\exists}\,, \dot{\forall}$ mapping propositional functions of type $\sigma \to o$ to propositions of type o.[3] If σ is an inductive set, it is possible to construct anomalous objects (that no longer fit in the intendend language) by making use of a case construct, e.g. $\dot{\forall}\,(\lambda x\!:\!\sigma.\, \mathsf{Case}\ x\ \mathsf{of}\ \ldots)$. Several possibilities have been explored to overcome this problem (apart from rejecting higher-order abstract syntax altogether), but many of them seem to harm the 'directness' of induction principles.

We choose the third option of representing variable bindings by the use of De Bruijn indices. The major advantage is that inductive definitions can be used in a direct way. The freely generated (structural) equality of inductively defined objects is the natural equality satisfying α-convertibility. Another advantage is the computational nature of the involved algorithms. Surely, there's more work for the programmer, but that's no reason not to do it.[4]

[2]A related problem is the conflict between higher-order abstract syntax and inductive definitions. A constructor of type $(o \to o) \to o$ cannot be accepted in an inductive definition, because of the negative (leftmost) occurrence of o. This problem is absent in the case of representing a first-order language.

[3]There, first-order terms are shallowly embedded, the domain of discourse σ, being a parameter set.

[4]On the contrary, Coq is the best game in town; it's fun!

The idea is to get rid of names altogether and replace a variable occurrence by a pointer to the corresponding binder. A variable is represented by a natural number which indicates the number of binders between the variable and its binder. For example, $\dot\forall\,\dot\exists\,\phi(v_1, v_0)$ reads as $\forall x.\, \exists y.\, \phi(x, y)$.[5]

Analytic versus synthetic judgements

Another design choice to be made is whether to localize derivations themselves. In the terminology of Martin-Löf, this is the distinction between *analytic* and *synthetic* judgements. Synthetic judgements are of the form $\Gamma \vdash \phi$ as opposed to analytic judgements $\Gamma \vdash d : \phi$, which carry their own evidence d. Objects d can be seen as λ-terms and formulas ϕ as classifying those λ-terms. Given our objective of building a 'tool' for manipulation of first-order proofs, the choice for analytic judgements is obvious. The advantage of analytic judgements is that we get more control over proofs and that such judgements are decidable, as will be shown in the sequel. We are able to perform computational proofs of lemma's about judgements, because instead of induction over a logical hypothesis $\Gamma \vdash \phi$, we can use structural recursion on a proof object. It has to be noted that, in the case of synthetic judgements, it's possible to view the constructors of $\vdash$ as constituting a λ-calculus. But those constructors have Γ and ϕ as arguments, which make them less practical to reason about or to manipulate. Throughout the paper we are somewhat loose in our use of syntax; application is sometimes written $(f\ x_0\ \ldots\ x_n)$, sometimes $f(x_0, \ldots, x_n)$.

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. Given our objective of building a 'tool' for proof manipulation, multiple sorts are not that interesting and we therefore freed ourselves of the technical care they would demand. However, as is well-known, 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 start by giving some preliminary definitions.

Definition 2.1 *Given a set σ and a list $L : (list\ \sigma)$, its* index set I_L *is defined by the equations:*

$$I_{[\,]} = \emptyset \qquad I_{[s|L']} = \mathbb{1} + I_{L'}$$

where $\emptyset$ is the empty set (i.e. without contructors), $\mathbb{1}$ the unit set (with one sole inhabitant) and $A + B$ is the disjoint sum of sets A and B. We write $|L|$ to denote the length of L. For the sake of readability we set $I_L = \{0, \ldots, |L| - 1\}$. Furthermore, we write $L(i)$ for the element indexed by $i : I_L$.

[5]De Bruijn counts from 1, we start counting from 0, consistent with the definition of $\mathbb{N}$.

Terms

Assume a list of natural numbers, representing function arities.

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

Definition 2.2 *The set τ of syntactic objects representing* first-order terms *is inductively defined by:*

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

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

Formulas

We assume a second list of natural numbers, representing relation arities.

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

Definition 2.3 *The set of objects o representing* predicate logical formulas, *is defined by the following abstract syntax, where $i : I_{l_{rel}}$, $k = l_{rel}(i)$ and $\phi, \psi : o$.*

$$o := \dot{\top} \mid \dot{\bot} \mid R_i(t_1, \ldots, t_k) \mid \phi \mathrel{\dot{\to}} \psi \mid \phi \mathrel{\dot{\wedge}} \psi \mid \phi \mathrel{\dot{\vee}} \psi \mid \dot{\forall}\, \phi \mid \dot{\exists}\, \phi$$

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

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

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

Derivations

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

Definition 2.4 *The syntactic class π of* proof terms *is defined by the grammar:*

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

where $n : \mathbb{N}$, $d, e, f : \pi$, $\phi : o$ and $t : \tau$.

As an example, we depict the construction $\vee^-(d, e, f, \phi)$ in traditional ND-format:

$$\dfrac{\begin{array}{ccc} & [\psi_1] & [\psi_2] \\ \vdots\ (d) & \vdots\ (e) & \vdots\ (f) \\ \psi_1 \vee \psi_2 & \phi & \phi \end{array}}{\phi}\ \vee\text{-elim}$$

Note that some constructors carry an argument of type o. Some constructors ($\bot^-$, $\to^+$, $\vee_l^+$, $\vee_r^+$, $\exists^+$ and $\exists^-$) carry such an argument in order to have proof terms uniquely determine natural deductions, as will be shown in the sequel (see Section 9). Would we have omitted the formula argument in e.g. $\to^+$, 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. Note that $\vee^-$ carries the formula it proves, although it can be inferred from its subproofs; this was done in order make the definition of one of the permutative conversions (rule PC$\exists\vee$ in Definition 13.1) easier; see Section 13.

In Section 5 we give an example of a proof term, and demonstrate that it 'inhabits' a certain tautology according to the deduction system defined in Section 4. First, we have to define substitution and some related operations.

3 Lifting and substitution

The representation of variables by De Bruijn indices requires an extra operation called *lifting*.[6] Lifting increments the *free* variables in a formula. Variables can only be 'grasped' if we know at what binding depth they reside. Therefore, functions dealing with terms (and formulas and proof terms) need an additional argument to store the binding depth.

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

Definition 3.1 *We define* term lifting $\uparrow_n t$ *by structural recursion on* $t : \tau$, *where* $n : \mathbb{N}$ *is the so-called binding depth. The first n variables,* $v_0, \ldots, v_{n-1}$, *are assumed to be bound (this information being imported from functions calling* $\uparrow_n t$*) and remain unchanged.*

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

We write $\uparrow t$ *to denote the lifting of all variables in t, shorthand for* $\uparrow_0 t$.

We need iterated lifting for the definition of substitution.[7]

Definition 3.2 Iterated term lifting.

$$\begin{array}{rcl} \uparrow_n^0 t & = & t \\ \uparrow_n^{m+1} t & = & \uparrow_n^m(\uparrow_n t) \end{array}$$

Definition 3.3 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 depth count, 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*

[6]In the literature on explicit substitutions (e.g. [BB+96]) 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 $\uparrow_n$ actually corresponds to $\Uparrow^n(\uparrow)$.

[7]This is only needed for Definition 3.3, *not* for the actual implementation.

variable v_n.

$$\begin{aligned} v_i[t]^n &= \begin{cases} v_i & \text{if } i<n \\ \uparrow_0^n t & \text{if } i=n \\ v_{i-1} & \text{if } i>n \end{cases} \\ f_i(t_1,\ldots,t_k)[t]^n &= f_i(t_1[t]^n,\ldots,t_k[t]^n) \end{aligned}$$

We set $t[t'] = t[t']^0$.

As aforementioned, the side-conditions (if $i < n$ etc.) in the above definitions are inefficient. As the unfolding of definitions proceeds, the side-conditions are left as proof obligations and their number increases exponentially. The implemented lifting and substitution functions are defined recursively and have no side-conditions (so that $\beta\delta\iota$-reduction does the job); $\uparrow_n t$ is encoded as $\mathit{lift\_trm}(n,t)$ and $t[t']^n$ is encoded as $\mathit{subst\_trm}(n,t,t')$.[8]

$$\begin{aligned} \mathit{lift}(0,i) &= i+1 \\ \mathit{lift}(n+1,0) &= 0 \\ \mathit{lift}(n+1,i+1) &= \mathit{lift}(n,i)+1 \end{aligned}$$

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

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

$$\begin{aligned} \mathit{subst\_trm}(n,v_i,t) &= \mathit{subst}(n,i,t) \\ \mathit{subst\_trm}(n,f_j(t_1,\ldots,t_k),t) &= f_j(\mathit{subst\_trm}(n,t_1,t),\ldots,\mathit{subst\_trm}(n,t_k,t)) \end{aligned}$$

Next we define the lifting and substitution operations on formulas.

Definition 3.4 The lifting of $\phi : o$ for binding depth n, *notation $\uparrow_n\phi$, is recursively defined as follows. Note the increment of the depth counter when a quantifier is passed.*

$$\begin{aligned} \uparrow_n\dot{\top} &= \dot{\top} \\ \uparrow_n\dot{\bot} &= \dot{\bot} \\ \uparrow_n R_i(t_1,\ldots,t_k) &= R_i(\uparrow_n t_1,\ldots,\uparrow_n t_k) \\ \uparrow_n(A \mathbin{\dot\wedge} B) &= \uparrow_n A \mathbin{\dot\wedge} \uparrow_n B \\ \uparrow_n(A \mathbin{\dot\vee} B) &= \uparrow_n A \mathbin{\dot\vee} \uparrow_n B \\ \uparrow_n(A \mathbin{\dot\to} B) &= \uparrow_n A \mathbin{\dot\to} \uparrow_n B \\ \uparrow_n(\dot\forall A) &= \dot\forall \uparrow_{n+1} A \\ \uparrow_n(\dot\exists A) &= \dot\exists \uparrow_{n+1} A \end{aligned}$$

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

[8] We found these definitions in [Per96], where they are attributed to Altenkirch [Alt94].

Definition 3.5 Substitution of $t : \tau$ for v_n[9] in $\phi : o$, *notation* $\phi[t]^n$, *is defined by recursion on the structure of* ϕ. *Again, as in Definition 3.4, n is the depth counter which increments when a quantifier is encountered during the recursive descending of the formula.*

$$\begin{array}{rcl}
\dot{\top}[t]^n & = & \dot{\top} \\
\dot{\bot}[t]^n & = & \dot{\bot} \\
R_i(t_1,\ldots,t_k)[t]^n & = & R_i(t_1[t]^n,\ldots,t_k[t]^n) \\
(\phi \dot{\to} \psi)[t]^n & = & \phi[t]^n \dot{\to} \psi[t]^n \\
(\phi \dot{\wedge} \psi)[t]^n & = & \phi[t]^n \dot{\wedge} \psi[t]^n \\
(\phi \dot{\vee} \psi)[t]^n & = & \phi[t]^n \dot{\vee} \psi[t]^n \\
(\dot{\forall}\,\phi)[t]^n & = & \dot{\forall}\,\phi[t]^{n+1} \\
(\dot{\exists}\,\phi)[t]^n & = & \dot{\exists}\,\phi[t]^{n+1}
\end{array}$$

We define $\phi[t] = \phi[t]^0$.

Note that, for the lifting and substitution operations, the same pattern repeats in similar cases: e.g. $\uparrow_n(A \circ B) = \uparrow_n A \circ \uparrow_n B$ for $\circ$ one of $\dot{\wedge}, \dot{\vee}, \dot{\to}$; or, even more general (binary connectives in prefix notation):

$$\begin{array}{rcl}
\uparrow_n(QM) & = & Q\uparrow_{n+1}M \text{ for } Q = \dot{\forall}, \dot{\exists} \\
\uparrow_n(MN) & = & \uparrow_n M \uparrow_n N \text{ for } M \text{ not a binder} \\
\uparrow_n \circ & = & \circ \text{ for } \circ = \top, \bot, R_i, \dot{\wedge}, \dot{\vee}, \dot{\to}
\end{array}$$

In the sequel (Section 12), we will encounter even more repetitions of the same pattern. The question of the implementational possibility of such a uniform and polymorphic (as there are three levels: τ,o and π) definition remains unanswered.

For the inference system introduced in the next section, we also need the lifting and substitution operation on *contexts*. Contexts are formalized as lists of formulas, written in reversed order.

Definition 3.6 Lifting of all free variables in context Γ, *given that the first n variables are bound, notation* $\uparrow_n\Gamma$, *is defined by:*

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

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

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

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

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

3.1 Free variables, finitely versus infinitely many

Note that, different 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 6.2). Alternatively, we could have chosen to parameterize 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 $\{0, \ldots, n\}$).

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

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, in order to be able to define a $\mathcal{V} : \mathbb{N} \to A$ for the evaluation of objects, one needs a default value in A.

3.2 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. However, as pointed out in [MP93], 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.

A function $\mathcal{V} : \mathbb{N} \to A$ (see the interpretation functions defined in Section 6) has to be taken into account. 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 e.g. $\mathcal{R}_i(x, y)$ into $\mathcal{R}_i(x, x)$, as illustrated below. Note that, for any $\mathcal{V}'$, if t is interpreted under $\mathcal{V}'$, then $t[t']$ has to be interpreted under $\uparrow_1^{\mathtt{v}} \mathcal{V}'$, 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 $\uparrow_1^{\mathtt{v}} \mathcal{V}(0) = \mathcal{V}(1) = x$ We have:

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

and

$$[\![R_i(v_1, v_0)[v_0]]\!]^{\uparrow_1^{\mathtt{v}} \mathcal{V}} = [\![R_i(v_0, v_0)]\!]^{\uparrow_1^{\mathtt{v}} \mathcal{V}} = \mathcal{R}_i(x, x)$$

However, we *cannot* make a substitution that transforms $\mathcal{R}_i(x, y)$ into $\mathcal{R}_i(y, y)$. 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 $\dot\exists\,\phi$. A third possible binder is the variable mapping $\mathcal{V}$ as exemplified above. We maintain the term "substitution" *par abus de langage*.

4 Judgements

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

DEFINITION 4.1 *The relation $(\Gamma \vdash d\ [:]\ \phi) : *^p$ is inductively defined by the following clauses. A context $\Gamma : (list\ o)$ is a list of formulas, as usual written in reversed order (i.e. the rightmost element has rank 0), $d, d_1, d_2, e_1, e_2 : \pi$ are proof terms, $t : \tau$ is a first-order term, and $\phi, \phi_1, \phi_2, \psi : o$ are formulas.*

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

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

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

$$\frac{\Gamma \vdash d_1\ [:]\ \phi_1 \quad \Gamma \vdash d_2\ [:]\ \phi_2}{\Gamma \vdash \wedge^+(d_1,d_2)\ [:]\ \phi_1 \mathbin{\dot\wedge} \phi_2} \qquad \frac{\Gamma \vdash d\ [:]\ \phi_1 \mathbin{\dot\wedge} \phi_2}{\Gamma \vdash \wedge_l^-(d)\ [:]\ \phi_1} \qquad \frac{\Gamma \vdash d\ [:]\ \phi_1 \mathbin{\dot\wedge} \phi_2}{\Gamma \vdash \wedge_r^-(d)\ [:]\ \phi_2}$$

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

$$\frac{\Gamma \vdash d\ [:]\ \psi_1 \mathbin{\dot\vee} \psi_2 \quad \Gamma;\psi_1 \vdash e_1\ [:]\ \phi \quad \Gamma;\psi_2 \vdash e_2\ [:]\ \phi}{\Gamma \vdash \vee^-(d,e_1,e_2,\phi)\ [:]\ \phi}$$

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

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

In contrast to a formalisation with named variables (see [Per96]), there is a canonical choice of a fresh variable in the setting with De Bruijn indices, as e.g. 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 v_0 becomes fresh.

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 which 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 16), e.g. the following formulation of

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

the rule for $\to^-$ is acceptable. Here proof terms d', e' are obtained from d, e respectively, by swapping and lifting of assumption variables so that they still refer to the same assumptions as they originally did. See sections 12 and 16 for these operations.

$$\frac{\Gamma \vdash d\ [:]\ \phi \mathrel{\dot\to} \psi \quad \Delta \vdash e\ [:]\ \phi}{\Gamma, \Delta \vdash \to^-(d', e')\ [:]\ \psi}$$

In Section 8 we show soundness of the deduction relation defined in 4.1, with respect to Coq's native logic. The next section illustrates the outlined constructions so far.

5 Example

Let $l_{rel} = [1, 1]$. This means that we have two unary predicates available, R_0 and R_1 both typed $\tau \to o$. Consider the following object $\theta : o$.[11]

$$\theta \equiv (\dot\exists\, R_0(v_0)) \mathbin{\dot\vee} (\dot\exists\, R_1(v_0)) \mathrel{\dot\to} \dot\exists\, R_1(v_0) \mathbin{\dot\vee} R_0(v_0)$$

With named variables this reads as follows.

$$(\exists x.\, R_0(x)) \vee (\exists y.\, R_1(y)) \to \exists z.\, R_1(z) \vee R_0(z)$$

We demonstrate how to construct an object $d : \pi$ such that $\vdash d\ [:]\ \theta$ holds.[12] We use the following abbreviations.

$$\begin{array}{lcl@{\qquad}lcl} \psi & \equiv & (\dot\exists\, R_0(v_0)) \mathbin{\dot\vee} (\dot\exists\, R_1(v_0)) & \Gamma_1 & \equiv & \psi; \dot\exists\, R_0(v_0) \\ \rho & \equiv & R_1(v_0) \mathbin{\dot\vee} R_0(v_0) & \Gamma_2 & \equiv & \psi; \dot\exists\, R_1(v_0) \end{array}$$

$$\dfrac{\Gamma_1 \vdash h_0\ [:]\ \dot\exists\, R_0(v_0) \quad \dfrac{\dfrac{\Gamma_1; R_0(v_0) \vdash h_0\ [:]\ R_0(v_0)}{\Gamma_1; R_0(v_0) \vdash \vee_r^+(R_1(v_0), h_0)\ [:]\ \rho}}{\Gamma_1; R_0(v_0) \vdash \exists^+(\rho, v_0, \vee_r^+(R_1(v_0), h_0))\ [:]\ \dot\exists\, \rho}}{\Gamma_1 \vdash \underbrace{\exists^-(h_0, \exists^+(\rho, v_0, \vee_r^+(R_1(v_0), h_0)), \dot\exists\, \rho)}_{d_1}\ [:]\ \dot\exists\, \rho}$$

$$\dfrac{\Gamma_2 \vdash h_0\ [:]\ \dot\exists\, R_1(v_0) \quad \dfrac{\dfrac{\Gamma_2; R_1(v_0) \vdash h_0\ [:]\ R_1(v_0)}{\Gamma_2; R_1(v_0) \vdash \vee_l^+(R_0(v_0), h_0)\ [:]\ \rho}}{\Gamma_2; R_1(v_0) \vdash \exists^+(\rho, v_0, \vee_l^+(R_0(v_0), h_0))\ [:]\ \dot\exists\, \rho}}{\Gamma_2 \vdash \underbrace{\exists^-(h_0, \exists^+(\rho, v_0, \vee_l^+(R_0(v_0), h_0))\dot\exists\, \rho)}_{d_2}\ [:]\ \dot\exists\, \rho}$$

Note that, as there are no free variables in $\Gamma_1, \Gamma_2, \dot\exists\, \rho$, we have that:

$$\uparrow\!\Gamma_1 = \Gamma_1 \quad \uparrow\!\Gamma_2 = \Gamma_2 \quad \uparrow\!(\dot\exists\, \rho) = \dot\exists\, \rho$$

and so both $\exists^-$ inferences are valid.

$$\dfrac{\dfrac{\psi \vdash h_0\ [:]\ \psi \quad \Gamma_1 \vdash d_1\ [:]\ \dot\exists\, \rho \quad \Gamma_2 \vdash d_2\ [:]\ \dot\exists\, \rho}{\psi \vdash \vee^-(h_0, d_1, d_2, \dot\exists\, \rho)\ [:]\ \dot\exists\, \rho}}{\vdash \to^+(\psi, \vee^-(h_0, d_1, d_2, \dot\exists\, \rho))\ [:]\ \psi \mathrel{\dot\to} \dot\exists\, \rho}$$

[11] As usual, we let $\dot\vee$ bind stronger than $\dot\to$. The scope of quantifiers extends to the right as far as brackets allow.

[12] Empty contexts are omitted.

6 Translation to **Coq**'s native logic

In this section we define the translation of object level statements (i.e. the objects defined in Definition 2.3) to meta-level statements (i.e. in the language of the framework itself). This translation will be referred to as interpretation.

First we introduce some operations on mappings $\mathcal{V} : \mathbb{N} \to A$ for some set A.

Definition 6.1 *Given $n : \mathbb{N}$, we define $\uparrow^{\mathrm{v}}_{n}\mathcal{V}$ as follows.*

$$\uparrow^{\mathrm{v}}_{0}\mathcal{V} = \mathcal{V} \qquad \uparrow^{\mathrm{v}}_{n+1}\mathcal{V} = \uparrow^{\mathrm{v}}_{n}\lambda p.\, \mathcal{V}(p+1)$$

For $n : \mathbb{N}$ and $a : A$, $\mathcal{V}[n := a]$ is defined as follows.

$$\mathcal{V}[0 := a](0) = a \qquad \mathcal{V}[0 := a](m+1) = \mathcal{V}(m)$$

$$\mathcal{V}[n+1 := a](0) = \mathcal{V}(0) \qquad \mathcal{V}[n+1 := a](m+1) = (\uparrow^{\mathrm{v}}_{1}\mathcal{V})[n := a](m)$$

Definition 6.2 *Assume an arbitrary domain of discourse $A : *^s$ and a function $\mathcal{V} : \mathbb{N} \to A$ to interpret (free) variables. Next we declare a parameter $\mathcal{F}$, a family of functions indexed over $I_{l_{fun}}$, used to interpret function symbols, where A^k is the cartesian product of k copies of A.*

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

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

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

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

$$\begin{array}{rcl} [\![\dot{\top}]\!]^{\mathcal{V}} & = & \top \\ [\![\dot{\bot}]\!]^{\mathcal{V}} & = & \bot \\ [\![R_i(t_1,\ldots,t_k)]\!]^{\mathcal{V}} & = & \mathcal{R}_i([\![t_1]\!]^{\mathcal{V}},\ldots,[\![t_k]\!]^{\mathcal{V}}) \\ [\![\phi \mathbin{\dot{\wedge}} \psi]\!]^{\mathcal{V}} & = & [\![\phi]\!]^{\mathcal{V}} \wedge [\![\psi]\!]^{\mathcal{V}} \\ [\![\phi \mathbin{\dot{\vee}} \psi]\!]^{\mathcal{V}} & = & [\![\phi]\!]^{\mathcal{V}} \vee [\![\psi]\!]^{\mathcal{V}} \\ [\![\phi \mathbin{\dot{\to}} \psi]\!]^{\mathcal{V}} & = & [\![\phi]\!]^{\mathcal{V}} \to [\![\psi]\!]^{\mathcal{V}} \\ [\![\dot{\forall}\, \phi]\!]^{\mathcal{V}} & = & \Pi x : A.\, [\![\phi]\!]^{\mathcal{V}[0:=x]} \\ [\![\dot{\exists}\, \phi]\!]^{\mathcal{V}} & = & \exists x : A.\, [\![\phi]\!]^{\mathcal{V}[0:=x]} \end{array}$$

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 isomorphism. We don't have to worry about name conflicts when adding 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 6.4 *The interpretation of a context is the conjunction of its interpreted elements.*

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

7 Thinning and substitution lemma's

It is possible to insert free variables to the mapping $\mathcal{V}$ of the interpretation function given in definitions 6.2, 6.3 and, if the argument is appropriately lifted, keep the same interpretations. This is called *thinning* and can be compared to *weakening* (see Lemma 16.2); the latter is about assumption variables, the former about term variables. First we define some auxiliary lemmas.

Lemma 7.1

$$(\mathcal{V}[n := x])[0 := y](m) = (\mathcal{V}[0 := y])[n+1 := x](m)$$

$$\uparrow^{\mathrm{v}}_{\mathcal{V}[0:=x]} n + 1(m) = \uparrow^{\mathcal{V}} n(m)$$

$$[\![\uparrow t]\!]^{\mathcal{V}} = [\![t]\!]^{\uparrow^{\mathrm{v}}_{1}\mathcal{V}}$$

For all $\mathcal{V}, \mathcal{V}' : \mathbb{N} \to A$ *and* $t : \tau$

$$(\Pi n.\, \mathcal{V}(n) = \mathcal{V}'(n)) \to [\![t]\!]^{\mathcal{V}} = [\![t]\!]^{\mathcal{V}'}$$

$$(\Pi n.\, \mathcal{V}(n) = \mathcal{V}'(n)) \to [\![p]\!]^{\mathcal{V}} \leftrightarrow [\![p]\!]^{\mathcal{V}'}$$

Lemma 7.2 *Thinning lemma. Let* $\mathcal{V} : \mathbb{N} \to A$, $a : A$ *and* $n : \mathbb{N}$.

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

The proof of the thinning lemma makes use of the following lemma's.

Similarly we need $[\![t[t']]\!]^{\mathcal{V}} = [\![t]\!]^{\mathcal{V}[0:=[\![t']\!]^{\mathcal{V}}]}$. We need induction loading, no longer assuming that $[\![t']\!]^{\mathcal{V}}$ is the last added element.

Lemma 7.3 *Substitution lemma.*

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

8 Soundness with respect to Coq's native logic

THEOREM 8.1 *For each $\Gamma : (list\ o)$, $d : \pi$, $\phi : o$ we have that:*

$$\Gamma \vdash d\ [:]\ \phi \to \Pi \mathcal{V} : \mathbb{N} \to A.\, [\![\Gamma]\!]^{\mathcal{V}} \to [\![\phi]\!]^{\mathcal{V}}$$

PROOF. By induction on the proposition $\Gamma \vdash d\ [:]\ \phi$. Quantification over $\mathcal{V} : \mathbb{N} \to A$ is necessary to get the appropriate induction hypotheses for cases $\forall^+$ and $\exists^-$. 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 following induction hypothesis.

$$IH : [\![\Gamma; \phi_1]\!]^{\mathcal{V}} \to [\![\phi_2]\!]^{\mathcal{V}}$$

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 applied to the pair[13] $\langle H_\Gamma, H_{\phi_1} \rangle$, is a proof of $[\![\phi_2]\!]^{\mathcal{V}}$.

Case $\Gamma \vdash \vee^-(d, e_1, e_2, \phi)\ [:]\ \phi$. Assume $H_\Gamma : [\![\Gamma]\!]^{\mathcal{V}}$. The proof obligation is $[\![\phi]\!]^{\mathcal{V}}$. We have three induction hypotheses, originating from the three premisses of the $\vee^-$-rule.

$$\begin{array}{rcl} IH_d & : & [\![\Gamma]\!]^{\mathcal{V}} \to [\![\psi_1 \dot{\vee} \psi_2]\!]^{\mathcal{V}} \\ IH_{e_1} & : & [\![\Gamma; \psi_1]\!]^{\mathcal{V}} \to [\![\phi]\!]^{\mathcal{V}} \\ IH_{e_2} & : & [\![\Gamma; \psi_2]\!]^{\mathcal{V}} \to [\![\phi]\!]^{\mathcal{V}} \end{array}$$

We get $[\![\psi_1]\!]^{\mathcal{V}} \vee [\![\psi_2]\!]^{\mathcal{V}}$ from IH_d and H_Γ.

- Suppose $H_{\psi_1} : [\![\psi_1]\!]^{\mathcal{V}}$, then $(IH_{e_1}\ \langle H_\Gamma, H_{\psi_1} \rangle) : [\![\phi]\!]^{\mathcal{V}}$.
- Suppose $H_{\psi_2} : [\![\psi_2]\!]^{\mathcal{V}}$, then $(IH_{e_2}\ \langle H_\Gamma, H_{\psi_2} \rangle) : [\![\phi]\!]^{\mathcal{V}}$.

Case $\Gamma \vdash \forall^+(d)\ [:]\ \dot{\forall} \phi$. Let $\mathcal{V} : \mathbb{N} \to A$ and $H_\Gamma : [\![\Gamma]\!]^{\mathcal{V}}$. We have:

$$IH_d : [\![\uparrow\Gamma]\!]^{\mathcal{V}} \to [\![\phi]\!]^{\mathcal{V}}$$

We have to prove:

$$\Pi x : A.\, [\![\phi]\!]^{\mathcal{V}[0:=x]}$$

Assume an arbitrary $x : A$. From Lemma 7.2 and H_Γ, it follows that $[\![\uparrow\Gamma]\!]^{\mathcal{V}[0:=x]}$. Then, IH_d for $\mathcal{V}[0 := x]$ and the proof of $[\![\uparrow\Gamma]\!]^{\mathcal{V}[0:=x]}$, proves $[\![\phi]\!]^{\mathcal{V}[0:=x]}$.

Case $\Gamma \vdash \exists^+(\phi, t, d)\ [:]\ \dot{\exists} \phi$. Let $H_\Gamma : [\![\Gamma]\!]^{\mathcal{V}}$. We have:

$$IH_d : [\![\Gamma]\!]^{\mathcal{V}} \to [\![\phi[t]]\!]^{\mathcal{V}}$$

[13] If $a : A$ and $b : B$, then $\langle a, b \rangle : A \wedge B$.

We have the following proof obligation.

$$\exists x\!:\!A.\, [\![\phi]\!]^{\mathcal{V}[0:=x]}$$

Give $[\![t]\!]^{\mathcal{V}}$ as witness for this existential statement, so that our goal becomes $[\![\phi]\!]^{\mathcal{V}[0:=[\![t]\!]^{\mathcal{V}}]}$, which—by Lemma 7.3—is implied by:

$$[\![\phi[t]]\!]^{\mathcal{V}}$$

which in turn follows directly from IH_d and H_Γ.

Until now, proof terms played no particular role. An analogous soundness result could have been obtained for judgements of the form $\Gamma \vdash \phi$. See the discussion on analytic versus synthetic judgements in the introduction. In the sequel it becomes clear how proof terms can be subject to manipulation.

9 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 synthesize the type of d.

First we define the set *opt* of so-called options; let $p : o$.

$$\mathit{opt} := \mathit{value}(p) \mid \mathit{error}$$

We define $\mathit{check}(\Gamma, d) : \mathit{opt}$ by recursion on d. 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 (decidable) boolean equality relation on formulas. If these conditions are not satisfied, *error* is returned. The canonical cases for the constructors $\wedge^+$, $\wedge^-_l$, $\wedge^-_r$, $\vee^+_l$, $\vee^+_r$ are left out.

Let us explain why the $\exists^-$-constructor expects an argument of type o.[14] In order to infer the type (an o-object) of a term $\exists^-(d, e)$, say ϕ, we need the type of the subterm e, say ϕ'. We know that ϕ' must be the lifted version of the formula we're looking for (if not, we are dealing with a term not corresponding to a correct deduction). So, we have to check that $\phi' = \uparrow\!\phi$ and that is why the $\exists^-$-constructor carries the extra argument ϕ. Consequently, the $\vee^-$-constructor

[14]There is an analogous explanation for the constructor $\exists^+$.

carries an o-argument too; otherwise we cannot define $\mapsto$ (see Section 13).

$$
\begin{array}{rcll}
\mathit{check}(\Gamma, \top^+) & = & \mathit{value}(\dot{\top}) & \\
\mathit{check}(\Gamma, h_i) & = & \mathit{value}(\Gamma(i)) & \text{if } i < |\Gamma| \\
\mathit{check}(\Gamma, \bot^-(d,\phi)) & = & \mathit{value}(\phi) & \text{if } \mathit{check}(\Gamma, d) = \mathit{value}(\dot{\bot}) \\
\mathit{check}(\Gamma, \to^+(\phi,d)) & = & \mathit{value}(\phi \dot{\to} \psi) & \text{if } \mathit{check}([\Gamma;\phi], d) = \mathit{value}(\psi) \\
\mathit{check}(\Gamma, \to^-(d,e)) & = & \mathit{value}(\psi) & \text{if } \begin{cases} \mathit{check}(\Gamma, d) = \mathit{value}(\phi \dot{\to} \psi) \\ \mathit{check}(\Gamma, e) = \mathit{value}(\phi') \\ \phi = \phi' \end{cases} \\
\mathit{check}(\Gamma, \vee^-(d,e_1,e_2,\phi)) & = & \mathit{value}(\phi) & \text{if } \begin{cases} \mathit{check}(\Gamma, d) = \mathit{value}(\rho_1 \mathbin{\dot{\vee}} \rho_2) \\ \mathit{check}([\Gamma;\rho_1], e_1) = \mathit{value}(\phi_1) \\ \mathit{check}([\Gamma;\rho_2], e_2) = \mathit{value}(\phi_2) \\ \phi_1 = \phi \\ \phi_2 = \phi \end{cases} \\
\mathit{check}(\Gamma, \forall^+(d)) & = & \mathit{value}(\dot{\forall}\, \phi) & \text{if } \mathit{check}(\uparrow\Gamma, d) = \mathit{value}(\phi) \\
\mathit{check}(\Gamma, \forall^-(t,d)) & = & \mathit{value}(\phi[t]) & \text{if } \mathit{check}(\Gamma, d) = \mathit{value}(\dot{\forall}\, \phi) \\
\mathit{check}(\Gamma, \exists^+(\phi,t,d)) & = & \mathit{value}(\dot{\exists}\, \phi) & \text{if } \begin{cases} \mathit{check}(\Gamma, d) = \mathit{value}(\phi') \\ \phi' = \phi[t] \end{cases} \\
\mathit{check}(\Gamma, \exists^-(d,e,\phi)) & = & \mathit{value}(\phi) & \text{if } \begin{cases} \mathit{check}(\Gamma, d) = \mathit{value}(\dot{\exists}\, \psi) \\ \mathit{check}([\uparrow\Gamma;\psi], e) = \mathit{value}(\phi') \\ \phi' = \uparrow\phi \end{cases}
\end{array}
$$

10 Correctness of *check*

THEOREM 10.1 *For all $d : \pi$, $\Gamma : (\mathit{list}\ o)$ and $\phi : o$, we have that:*

$$\mathit{check}(\Gamma, d) = \mathit{value}(\phi) \leftrightarrow \Gamma \vdash d\ [:]\ \phi$$

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

11 Unique types

Proof terms have unique types.

LEMMA 11.1 *For all contexts Γ, proof terms d and formulas ϕ, ψ, we have that:*

$$(\Gamma \vdash d\ [:]\ \phi) \to (\Gamma \vdash d\ [:]\ \psi) \to \phi = \psi$$

PROOF. Direct from double application of Theorem 10.1.

12 Lifting and substitution in proof terms

In this section we define the lifting and substitution operations in proof terms, both for term variables (of type τ), notations $\uparrow^{\mathrm{v}}_n d$ and $d[t]^n_{\mathrm{v}}$, as well as for assumption variables (of type π) , notations $\uparrow^{\mathrm{h}}_n d$ and $d[d']^n_{\mathrm{h}}$.

Lifting and substitution of term variables

Lifting and substitution in proof terms of term variables share the same recursive structure; the depth counter they carry increments in the cases of $\forall^+$, $\exists^+$ and $\exists^-$ (second argument). Instead of duplicating this structure, we abstract from what should happen to terms (g_1) and formulas (g_2) and define the function $\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, d)$.

Definition 12.1 *Given $g_1 : \mathbb{N} \to \tau \to \tau$ and $g_2 : \mathbb{N} \to o \to o$, the function $\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, d)$ is defined by the following recursive equations, where $n, i : \mathbb{N}$, $d, e_1, e_2 : \pi$, $t : \tau$ and $\phi : o$.*

$$\begin{array}{rcl}
\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, \top^+) & = & \top^+ \\
\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, \bot^-(d, \phi)) & = & \bot^-(\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, d), g2(n, \phi)) \\
\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, h_i) & = & h_i \\
\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, \to^+(\phi, d)) & = & \to^+(g_2(n, \phi), \mathit{rec}_{\mathrm{v}}(g_1, g_2, n, d)) \\
\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, \to^-(d, e)) & = & \to^-(\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, d), \mathit{rec}_{\mathrm{v}}(g_1, g_2, n, e)) \\
\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, \wedge^+(d, e)) & = & \wedge^+(\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, d), \mathit{rec}_{\mathrm{v}}(g_1, g_2, n, e)) \\
\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, \wedge_l^-(d)) & = & \wedge_l^-(\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, d)) \\
\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, \wedge_r^-(d)) & = & \wedge_r^-(\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, d)) \\
\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, \vee_l^+(\phi, d)) & = & \vee_l^+(g_2(n, \phi), \mathit{rec}_{\mathrm{v}}(g_1, g_2, n, d)) \\
\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, \vee_r^+(\phi, d)) & = & \vee_r^+(g_2(n, \phi), \mathit{rec}_{\mathrm{v}}(g_1, g_2, n, d)) \\
\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, \vee^-(d, e_1, e_2, \phi)) & = & \vee^-(\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, d), \mathit{rec}_{\mathrm{v}}(g_1, g_2, n, e_1), \mathit{rec}_{\mathrm{v}}(g_1, g_2, n, e_2), g_2(n, \phi)) \\
\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, \forall^+(d)) & = & \forall^+(\mathit{rec}_{\mathrm{v}}(g_1, g_2, n+1, d)) \\
\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, \forall^-(t, d)) & = & \forall^-(g_1(n, t), \mathit{rec}_{\mathrm{v}}(g_1, g_2, n, d)) \\
\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, \exists^+(\phi, t, d)) & = & \exists^+(g_2(n+1, \phi), g_1(n, t), \mathit{rec}_{\mathrm{v}}(g_1, g_2, n, d)) \\
\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, \exists^-(d, e, \phi)) & = & \exists^-(\mathit{rec}_{\mathrm{v}}(g_1, g_2, n, d), \mathit{rec}_{\mathrm{v}}(g_1, g_2, n+1, e), g_2(n, \phi))
\end{array}$$

The case of $\exists^+$ in Definition 12.1 may come as a surprise. Recall the corresponding inference rule given in Definition 4.1. The argument ϕ in term $\exists^+(\phi, t, d)$ has free variable v_0 ('from the outside'), so that it can be checked that subterm d is of type $\phi[t]$, i.e. the first free variable replaced by the witnessing t. This free variable should remain free. Therefore the depth counter is incremented, just like the cases of $\forall^+$ and $\exists^-$ (second argument).

Definition 12.2 *For $n : \mathbb{N}$ and $d : \pi$,* lifting of term variables in proof terms *$\uparrow_n^{\mathrm{v}} d$ is defined as follows.*

$$\uparrow_n^{\mathrm{v}} d = \mathit{rec}_{\mathrm{v}}(\lambda u.\, \lambda m.\, \uparrow_m u, \lambda \phi.\, \lambda m.\, \uparrow_m \phi, n, d)$$

Definition 12.3 *For $n : \mathbb{N}$, $t : \tau$ and $d : \pi$,* substitution of term variables in proof terms *$d[t]_{\mathrm{v}}^n$ is defined by:*

$$d[t]_{\mathrm{v}}^n = \mathit{rec}_{\mathrm{v}}(\lambda u.\, \lambda m.\, u[t]^m, \lambda \phi.\, \lambda m.\, \phi[t]^m, n, d)$$

Lifting and substitution of assumption variables

Several proof term transformations concerning assumption variables recursively descend in the same way. The reference depth of assumption variables is incremented in the cases of $\to^+$ (second argument), $\vee^-$ (second and third argument) and $\exists^-$ (second argument); i.e. any time an extra hypothesis is added to the context (the inference rules of $\vdash$ viewed bottom up). Instead of repeating the structure of these definitions, we abstract from the function which is applied in the case of h_i. The function rec_{h} defined below, will be reused four times: for lifting and substitution of assumption variables, as well as for the proof term transformations necessary for the admissable structural rules *exchange* and *contraction* (see Section 16).

Definition 12.4 *Let $g : \mathbb{N} \to \mathbb{N} \to \pi$, a function which returns a proof term given two natural numbers (depth counter, resp. index of assumption variable), $n : \mathbb{N}$, and $d : \pi$, then $rec_{\mathrm{h}}(g, n, d)$ is defined by the following recursive equations, where $i : \mathbb{N}$, $d, e_1, e_2 : \pi$, $t : \tau$ and $\phi : o$.*

$$\begin{array}{rcl}
rec_{\mathrm{h}}(g,n,\top^+) & = & \top^+ \\
rec_{\mathrm{h}}(g,n,\bot^-(d,\phi)) & = & \bot^-(rec_{\mathrm{h}}(g,n,d),\phi) \\
rec_{\mathrm{h}}(g,n,h_i) & = & g(n,i) \\
rec_{\mathrm{h}}(g,n,\to^+(\phi,d)) & = & \to^+(\phi, rec_{\mathrm{h}}(g,n+1,d)) \\
rec_{\mathrm{h}}(g,n,\to^-(d,e)) & = & \to^-(rec_{\mathrm{h}}(g,n,d), rec_{\mathrm{h}}(g,n,e)) \\
rec_{\mathrm{h}}(g,n,\wedge^+(d,e)) & = & \wedge^+(rec_{\mathrm{h}}(g,n,d), rec_{\mathrm{h}}(g,n,e)) \\
rec_{\mathrm{h}}(g,n,\wedge_l^-(d)) & = & \wedge_l^-(rec_{\mathrm{h}}(g,n,d)) \\
rec_{\mathrm{h}}(g,n,\wedge_r^-(d)) & = & \wedge_r^-(rec_{\mathrm{h}}(g,n,d)) \\
rec_{\mathrm{h}}(g,n,\vee_l^+(\phi,d)) & = & \vee_l^+(\phi, rec_{\mathrm{h}}(g,n,d)) \\
rec_{\mathrm{h}}(g,n,\vee_r^+(\phi,d)) & = & \vee_r^+(\phi, rec_{\mathrm{h}}(g,n,d)) \\
rec_{\mathrm{h}}(g,n,\vee^-(d,e_1,e_2,\phi)) & = & \vee^-(rec_{\mathrm{h}}(g,n,d), rec_{\mathrm{h}}(g,n+1,e_1), rec_{\mathrm{h}}(g,n+1,e_2),\phi) \\
rec_{\mathrm{h}}(g,n,\forall^+(d)) & = & \forall^+(rec_{\mathrm{h}}(g,n,d)) \\
rec_{\mathrm{h}}(g,n,\forall^-(t,d)) & = & \forall^-(t, rec_{\mathrm{h}}(g,n,d)) \\
rec_{\mathrm{h}}(g,n,\exists^+(\phi,t,d)) & = & \exists^+(\phi,t, rec_{\mathrm{h}}(g,n,d)) \\
rec_{\mathrm{h}}(g,n,\exists^-(d,e,\phi)) & = & \exists^-(rec_{\mathrm{h}}(g,n,d), rec_{\mathrm{h}}(g,n+1,e),\phi)
\end{array}$$

Definition 12.5 Lifting of assumption variables in proof terms *is defined by*

$$\uparrow_n^{\mathrm{h}} d = rec_{\mathrm{h}}(\lambda m.\, \lambda i.\, h_{lift(m,i)}, n, d)$$

The function lift is defined in Section 3. Define $\uparrow^{\mathrm{h}} d = \uparrow_0^{\mathrm{h}} d$. The definition of substitution below requires the definition of iterated lifting.

$$\begin{array}{rcl}
\uparrow_n^0 d & = & d \\
\uparrow_n^{m+1} d & = & \uparrow_n^m(\uparrow_n^{\mathrm{h}} d)
\end{array}$$

Definition 12.6 Substitution of proof terms for assumption variables in proof terms *is defined by*

$$d[d']_{\mathrm{h}}^n = rec_{\mathrm{h}}(\lambda m.\, \lambda i.\, h_i[d']_{\mathrm{h}}^m, n, d)$$

where

$$h_i[d']_{\mathrm{h}}^n = \begin{cases} h_i & \text{if } i < n \\ \uparrow_0^n d' & \text{if } i = n \\ h_{i-1} & \text{if } i > n \end{cases}$$

It should be noted that $h_i[d']_{\mathrm{h}}^n$ is encoded without side-conditions, in a similar way as $v_i[t]^n$ (see Definition 3.3). Define $d[d']_{\mathrm{h}} = d[d']_{\mathrm{h}}^0$.

13 Proof reduction

To illustrate how the defined machinery can be used to manipulate proof objects, we define Prawitz' proof reduction rules [Pra71].[15] The goal is to remove detours, as in the following tree.

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

Instead of first assuming ϕ to build a proof d of ψ, introduce the implication $\phi \dot{\rightarrow} \psi$ 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.

$$\dfrac{}{\Gamma \vdash d[e]_{\mathrm{h}} \;[:]\; \psi}$$

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 17.1 demonstrates why the various lifting operations are necessary to keep correct proofs.

Definition 13.1 Immediate proof reduction *is defined by the following rewrite rules. The left hand sides are called (proper, permutative) redexes and the right hand sides immediate (proper, permutative) reducts.*
Proper reductions.

$$\begin{array}{rcll}
\rightarrow^{-}(\rightarrow^{+}(\phi, d), e) & \mapsto & d[e]_{\mathrm{h}} & (\mathrm{PR}_{\rightarrow}) \\
\wedge_{l}^{-}(\wedge^{+}(d_1, d_2)) & \mapsto & d_1 & (\mathrm{PR}\wedge_1) \\
\wedge_{r}^{-}(\wedge^{+}(d_1, d_2)) & \mapsto & d_2 & (\mathrm{PR}\wedge_2) \\
\vee^{-}(\vee_{l}^{+}(\phi, d), e_1, e_2, \psi) & \mapsto & e_1[d]_{\mathrm{h}} & (\mathrm{PR}\vee_1) \\
\vee^{-}(\vee_{r}^{+}(\phi, d), e_1, e_2, \psi) & \mapsto & e_2[d]_{\mathrm{h}} & (\mathrm{PR}\vee_2) \\
\forall^{-}(t, \forall^{+}(d)) & \mapsto & d[t]_{\mathrm{v}} & (\mathrm{PR}\forall) \\
\exists^{-}(\exists^{+}(\phi, t, d), e) & \mapsto & (e[t]_{\mathrm{v}})[d]_{\mathrm{h}} & (\mathrm{PR}\exists)
\end{array}$$

[15]We actually follow [Pol96], pages 85–88.

Permutative conversions.

$$\begin{aligned}
\bot^-(\vee^-(d, e_1, e_2, \dot{\bot}), \phi) &\mapsto \vee^-(d, \bot^-(e_1, \phi), \bot^-(e_2, \phi), \phi) && (\mathrm{PC}\vee\bot)\\
\to^-(\vee^-(d, e_1, e_2, \phi \mathbin{\dot{\to}} \psi), g) &\mapsto \vee^-(d, \to^-(e_1, \uparrow^{\mathrm{h}} g), \to^-(e_2, \uparrow^{\mathrm{h}} g), \psi) && (\mathrm{PC}\vee\to)\\
\wedge_l^-(\vee^-(d, e_1, e_2, \phi \mathbin{\dot{\wedge}} \psi)) &\mapsto \vee^-(d, \wedge_l^-(e_1), \wedge_l^-(e_2), \phi) && (\mathrm{PC}\vee\wedge_1)\\
\wedge_r^-(\vee^-(d, e_1, e_2, \phi \mathbin{\dot{\wedge}} \psi)) &\mapsto \vee^-(d, \wedge_r^-(e_1), \wedge_r^-(e_2), \psi) && (\mathrm{PC}\vee\wedge_2)\\
\vee^-(\vee^-(d, e_1, e_2, \phi_1 \mathbin{\dot{\vee}} \phi_2), g, h, \psi) &\mapsto \vee^-(d, \vee^-(e_1, \uparrow_1^{\mathrm{h}} g, \uparrow_1^{\mathrm{h}} h, \psi), \vee^-(e_2, \uparrow_1^{\mathrm{h}} g, \uparrow_1^{\mathrm{h}} h, \psi), \psi) && (\mathrm{PC}\vee\vee)\\
\forall^-(t, \vee^-(d, e_1, e_2, \dot{\forall}\, \phi)) &\mapsto \vee^-(d, \forall^-(t, e_1), \forall^-(t, e_2), \phi[t]) && (\mathrm{PC}\vee\forall)\\
\exists^-(\vee^-(d, e_1, e_2, \dot{\exists}\, \phi), g, \psi) &\mapsto \vee^-(d, \exists^-(e_1, \uparrow_1^{\mathrm{h}} g, \psi), \exists^-(e_2, \uparrow_1^{\mathrm{h}} g, \psi), \psi) && (\mathrm{PC}\vee\exists)\\
\bot^-(\exists^-(d, e, \dot{\bot}), \phi) &\mapsto \exists^-(d, \bot^-(e, \uparrow\phi), \phi) && (\mathrm{PC}\exists\bot)\\
\to^-(\exists^-(d, e, \phi \mathbin{\dot{\to}} \psi), f) &\mapsto \exists^-(d, \to^-(e, \uparrow^{\mathrm{h}}(\uparrow^{\mathrm{v}} f)), \psi) && (\mathrm{PC}\exists\to)\\
\wedge_l^-(\exists^-(d, e, \phi \mathbin{\dot{\wedge}} \psi)) &\mapsto \exists^-(d, \wedge_l^-(e), \phi) && (\mathrm{PC}\exists\wedge_1)\\
\wedge_r^-(\exists^-(d, e, \phi \mathbin{\dot{\wedge}} \psi)) &\mapsto \exists^-(d, \wedge_r^-(e), \psi) && (\mathrm{PC}\exists\wedge_2)\\
\vee^-(\exists^-(d, e, \phi_1 \mathbin{\dot{\vee}} \phi_2), f, g, \psi) &\mapsto \exists^-(d, \vee^-(e, \uparrow_1^{\mathrm{h}}(\uparrow^{\mathrm{v}} f), \uparrow_1^{\mathrm{h}}(\uparrow^{\mathrm{v}} g), \uparrow\psi), \psi) && (\mathrm{PC}\exists\vee)\\
\forall^-(t, \exists^-(d, e, \dot{\forall}\, \phi)) &\mapsto \exists^-(d, \forall^-(\uparrow t, e), p[t]) && (\mathrm{PC}\exists\forall)\\
\exists^-(\exists^-(d, e, \dot{\exists}\, \phi), f, \psi) &\mapsto \exists^-(d, \exists^-(e, \uparrow_1^{\mathrm{h}}(\uparrow_1^{\mathrm{v}} f), \uparrow\psi), \psi) && (\mathrm{PC}\exists\exists)
\end{aligned}$$

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

$$\begin{aligned}
&\exists^-(\vee^-(d, \exists^+(\phi, t, e_1), e_2, \dot{\exists}\, \phi), g, \psi)\\
&\mapsto \vee^-(d, \exists^-(\exists^+(\phi, t, e_1), \uparrow_1^{\mathrm{h}} g, \psi), \exists^-(e_2, \uparrow_1^{\mathrm{h}} g, \psi), \psi)\\
&\mapsto \vee^-(d, ((\uparrow_1^{\mathrm{h}} g)[t]_{\mathrm{v}})[e_1]_{\mathrm{h}}, \exists^-(e_2, \uparrow_1^{\mathrm{h}} g, \psi))
\end{aligned}$$

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

$$\dfrac{\dfrac{\Gamma \vdash d\ [:]\ \rho_1 \mathbin{\dot{\vee}} \rho_2 \quad \dfrac{\Gamma; \rho_1 \vdash e_1\ [:]\ \phi[t]}{\Gamma; \rho_1 \vdash \exists^+(\phi, t, e_1)\ [:]\ \dot{\exists}\, \phi} \quad \Gamma; \rho_2 \vdash e_2\ [:]\ \dot{\exists}\, \phi}{\Gamma \vdash \vee^-(d, \exists^+(\phi, t, e_1), e_2, \dot{\exists}\, \phi)\ [:]\ \dot{\exists}\, \phi} \qquad \uparrow\Gamma; \phi \vdash g\ [:]\ \uparrow\psi}{\Gamma \vdash \exists^-(\vee^-(d, \exists^+(\phi, t, e_1), e_2, \dot{\exists}\, \phi), g, \psi)\ [:]\ \psi}$$

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

$$\dfrac{\Gamma \vdash d\ [:]\ \rho_1 \mathbin{\dot{\vee}} \rho_2 \quad \mathcal{T}_1 \quad \mathcal{T}_2}{\Gamma \vdash \vee^-(d, \exists^-(\exists^+(\phi, t, e_1), \uparrow_1^{\mathrm{h}} g, \psi), \exists^-(e_2, \uparrow_1^{\mathrm{h}} g, \psi), \psi)\ [:]\ \psi}$$

Where $\mathcal{T}_1$ denotes

$$\dfrac{\Gamma; \rho_1 \vdash \exists^+(\phi, t, e_1)\ [:]\ \dot{\exists}\, \phi \quad \uparrow(\Gamma; \rho_1); \phi \vdash \uparrow_1^{\mathrm{h}} g\ [:]\ \uparrow\psi}{\Gamma; \rho_1 \vdash \exists^-(\exists^+(\phi, t, e_1), \uparrow_1^{\mathrm{h}} g, \psi)\ [:]\ \psi}$$

and $\mathcal{T}_2$ denotes

$$\dfrac{\Gamma; \rho_2 \vdash e_2\ [:]\ \dot{\exists}\, \phi \quad \uparrow(\Gamma; \rho_2); \phi \vdash \uparrow_1^{\mathrm{h}} g\ [:]\ \uparrow\psi}{\Gamma; \rho_2 \vdash \exists^-(e_2, \uparrow_1^{\mathrm{h}} g, \psi)\ [:]\ \psi}$$

Now $\mathcal{T}_1$ contains a direct detour, which reduces to:

$$\dfrac{}{\Gamma; \rho_1 \vdash ((\uparrow_1^{\mathrm{h}} g)[t]_{\mathrm{v}})[e_1]_{\mathrm{h}}\ [:]\ \psi}$$

Definition 13.2 *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.*

14 Properties of De Bruijn operations

We present some basic algebraic properties of the operations introduced in Section 3. Similar properties can be found in [BW97].[16] All five lemma's are proved for both $t : \tau$ as well as $t : o$. Furthermore $t', t_1, t_2 : \tau$ and $n, m : \mathbb{N}$.

Lemma 14.1 Permutation of lifting.

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

Lemma 14.2 Simplification of substitution.

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

Lemma 14.3 Commutation of lifting and substitution. *Proof uses Lemma 14.1.*

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

Lemma 14.4 Distribution of lifting over substitution. *Proof uses Lemma 14.1.*

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

Lemma 14.5 Distribution of substitution. *Proof uses Lemma 14.3.*

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

15 Inversion lemma's

Derivation trees (Def. 4.1) and derivation terms (Def. 2.4) have the same shape. Given a judgement $\Gamma \vdash d$ [:] ϕ, the structure of the proof term d tells us how the judgement must have been derived. The inversion lemma's state that the inference rules of $\vdash$ can be inverted. Consequently, a subterm of a well-typed term is well-typed.

Coq provides a tactic `Inversion` for such situations. This tactic applied to a term of type $(P\ \vec{t})$, where P is an inductive predicate, derives for each possible constructor c_i of $(P\ \vec{t})$ the necessary conditions that should hold for the instance $(P\ \vec{t})$ to be proved by c_i. Although this tactic is very useful, the proof terms it builds are relatively large, which is the reason to use the following lemma's (proved by using the `Inversion` tactic) instead, and use them as opaque constants in other proofs (e.g. in the proof of Subject Reduction, see Section 17).[17]

[16]Where they are attributed to [Hue93].

[17]We experimented with both options; one proof of SR using the `Inversion` tactic, and another using the inversion lemma's listed here. Once compiled, the latter proof is a factor 7 smaller than the former one.

LEMMA 15.1

$$
\begin{array}{rcl}
\vdash h_i \;[:]\; \phi & \rightarrow & \bot \\
\Gamma; \psi \vdash h_0 \;[:]\; \phi & \rightarrow & \phi = \psi \\
\Gamma; \psi \vdash h_{i+1} \;[:]\; \phi & \rightarrow & \Gamma \vdash h_i \;[:]\; \phi \\
\Gamma \vdash \top^+ \;[:]\; \phi & \rightarrow & \phi = \top \\
\Gamma \vdash \bot^-(d, \psi) \;[:]\; \phi & \rightarrow & \left\{ \begin{array}{l} \phi = \psi \\ \Gamma \vdash d \;[:]\; \bot \end{array} \right. \\
\Gamma \vdash \rightarrow^+(\psi_1, d) \;[:]\; \phi & \rightarrow & \exists \psi_2 : o. \left\{ \begin{array}{l} \phi = \psi_1 \mathrel{\dot{\rightarrow}} \psi_2 \\ \Gamma; \psi_1 \vdash d \;[:]\; \psi_2 \end{array} \right. \\
\Gamma \vdash \rightarrow^-(d, e) \;[:]\; \psi & \rightarrow & \exists \phi : o. \left\{ \begin{array}{l} \Gamma \vdash d \;[:]\; \phi \mathrel{\dot{\rightarrow}} \psi \\ \Gamma \vdash e \;[:]\; \phi \end{array} \right. \\
\Gamma \vdash \wedge^+(d_1, d_2) \;[:]\; \phi & \rightarrow & \exists \phi_1, \phi_2 : o. \left\{ \begin{array}{l} \phi = \phi_1 \mathbin{\dot{\wedge}} \phi_2 \\ \Gamma \vdash d_1 \;[:]\; \phi_1 \\ \Gamma \vdash d_2 \;[:]\; \phi_2 \end{array} \right. \\
\Gamma \vdash \wedge_l^-(d) \;[:]\; \phi & \rightarrow & \exists \psi : o.\, \Gamma \vdash d \;[:]\; \phi \mathbin{\dot{\wedge}} \psi \\
\Gamma \vdash \wedge_r^-(d) \;[:]\; \psi & \rightarrow & \exists \phi : o.\, \Gamma \vdash d \;[:]\; \phi \mathbin{\dot{\wedge}} \psi \\
\Gamma \vdash \vee_l^+(\phi_2, d) \;[:]\; \phi & \rightarrow & \exists \phi_1 : o. \left\{ \begin{array}{l} \phi = \phi_1 \mathbin{\dot{\vee}} \phi_2 \\ \Gamma \vdash d \;[:]\; \phi_1 \end{array} \right. \\
\Gamma \vdash \vee_r^+(\phi_1, d) \;[:]\; \phi & \rightarrow & \exists \phi_2 : o. \left\{ \begin{array}{l} \phi = \phi_1 \mathbin{\dot{\vee}} \phi_2 \\ \Gamma \vdash d \;[:]\; \phi_2 \end{array} \right. \\
\Gamma \vdash \vee^-(d, e_1, e_2, \phi) \;[:]\; \phi & \rightarrow & \exists \psi_1, \psi_2 : o. \left\{ \begin{array}{l} \Gamma \vdash d \;[:]\; \psi_1 \mathbin{\dot{\vee}} \psi_2 \\ \Gamma; \psi_1 \vdash e_1 \;[:]\; \phi \\ \Gamma; \psi_2 \vdash e_2 \;[:]\; \phi \end{array} \right. \\
\Gamma \vdash \forall^+(d) \;[:]\; \phi & \rightarrow & \exists \phi' : o. \left\{ \begin{array}{l} \phi = \dot{\forall}\, \phi' \\ \uparrow\Gamma \vdash d \;[:]\; \phi' \end{array} \right. \\
\Gamma \vdash \forall^-(t, d) \;[:]\; \phi & \rightarrow & \exists \phi' : o. \left\{ \begin{array}{l} \phi = \phi'[t] \\ \Gamma \vdash d \;[:]\; \dot{\forall}\, \phi' \end{array} \right. \\
\Gamma \vdash \exists^+(\phi', t, d) \;[:]\; \phi & \rightarrow & \left\{ \begin{array}{l} \phi = \dot{\exists}\, \phi' \\ \Gamma \vdash d \;[:]\; \phi'[t] \end{array} \right. \\
\Gamma \vdash \exists^-(d, e, \phi) \;[:]\; \phi & \rightarrow & \exists \psi : o. \left\{ \begin{array}{l} \Gamma \vdash d \;[:]\; \dot{\exists}\, \psi \\ \uparrow\Gamma; \psi \vdash e \;[:]\; \uparrow\phi \end{array} \right.
\end{array}
$$

16 Admissable rules

The following rules are *admissable*, i.e. derivable in the meta-theory. They are used in the proof of Theorem 17.2. In order to prove by induction, the statements are loaded appropriately, i.e. quantify over Γ, Δ, etc.

LEMMA 16.1 *Lifting of judgement.*

$$\frac{\Gamma \vdash d \;[:]\; \phi}{\uparrow_n\Gamma \vdash \uparrow_n^{\mathrm{v}} d \;[:]\; \uparrow_n\phi}$$

PROOF. Induction on the proposition $\Gamma \vdash d \;[:]\; \phi$. The proofs of cases $\forall^+$ and $\exists^-$ require Lemma 14.1; cases $\forall^-$ and $\exists^+$ require Lemma 14.4.

LEMMA 16.2 *Weakening. Analogous to Lemma 7.2.*

$$\frac{\Gamma; \Delta \vdash d \;[:]\; \phi}{\Gamma; \psi; \Delta \vdash \uparrow_{|\Delta|}^{\mathrm{h}} d \;[:]\; \phi}$$

PROOF. By induction on d and inversion lemma's.

LEMMA 16.3 *Substitution of variables v_i in derivation terms.*

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

PROOF. By induction on d and inversion lemma's. Case h_i is proved by induction over i and Lemma 14.2. Cases $\forall^+$ and $\exists^-$ require lemma's 14.3 and 14.1. Cases $\forall^-$ and $\exists^-$ require Lemma 14.5.

LEMMA 16.4 *Substitution of variables h_i in derivation terms. Analogous to Lemma 7.3.*

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

PROOF. By induction on e and inversion lemma's. Case h_i is proved by induction over i and Lemma 16.2.

Exchange, contraction

The structural rules *exchange* and *contraction* are admissable too.[18] 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 lemma's 16.5 and 16.6).

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

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

LEMMA 16.5 *Exchange.*

$$\frac{\Gamma;\psi;\phi;\Delta \vdash d\ [:]\ \rho}{\Gamma;\phi;\psi;\Delta \vdash rec_{\mathrm{h}}(exch,|\Delta|,d)\ [:]\ \rho}$$

LEMMA 16.6 *Contraction.*

$$\frac{\Gamma;\phi;\phi;\Delta \vdash d\ [:]\ \psi}{\Gamma;\phi;\Delta \vdash rec_{\mathrm{h}}(contr,|\Delta|,d)\ [:]\ \psi}$$

17 Subject reduction

THEOREM 17.1

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

PROOF. By induction on the proposition $d \mapsto e$. The so obtained instances of $\Gamma \vdash d\ [:]\ \phi$ are inverted twice, using the inversion lemma's given in 15.1. We show some representative cases.

[18]These lemma's are not needed in the proof of Subject Reduction (Thm. 17.2).

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

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

The proof obligation is $\Gamma \vdash d[e]_{\mathrm{h}}$ [:] ψ, which follows from Lemma 16.4 by substituting the empty context for Δ:

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

PR$\forall$ 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: $\Gamma \vdash d[t]_{\mathrm{v}}$ [:] $\phi[t]$, which follows from Lemma 16.3 and $\uparrow\Gamma \vdash d$ [:] ϕ (take Δ empty and $n = 0$).

PC$\vee\vee$ Assume $\Gamma \vdash \vee^-(\vee^-(d, e_1, e_2, \psi_1 \dot\vee \psi_2), g, h, \phi)$ [:] ϕ. The proof obligation is:

$$\Gamma \vdash \vee^-(d, \vee^-(e_1, \uparrow^{\mathrm{h}}_1 g, \uparrow^{\mathrm{h}}_1 h), \vee^-(e_2, \uparrow^{\mathrm{h}}_1 g, \uparrow^{\mathrm{h}}_1 h)) \text{ [:] } \phi$$

We use the following abbreviations.

$$\begin{array}{lcl} J_d & \equiv & \Gamma \vdash d \text{ [:] } \rho_1 \dot\vee \rho_2 \\ J_g & \equiv & \Gamma; \psi_1 \vdash g \text{ [:] } \phi \\ J_h & \equiv & \Gamma; \psi_2 \vdash h \text{ [:] } \phi \end{array} \qquad \begin{array}{lcl} J_{e_1} & \equiv & \Gamma; \rho_1 \vdash e_1 \text{ [:] } \psi_1 \dot\vee \psi_2 \\ J_{e_2} & \equiv & \Gamma; \rho_2 \vdash e_2 \text{ [:] } \psi_1 \dot\vee \psi_2 \end{array}$$

After inversion, we come to the following tree.

$$\dfrac{\dfrac{J_d \quad J_{e_1} \quad J_{e_2}}{\Gamma \vdash \vee^-(d, e_1, e_2, \psi_1 \dot\vee \psi_2) \text{ [:] } \psi_1 \dot\vee \psi_2} \quad J_g \quad J_h}{\Gamma \vdash \vee^-(\vee^-(d, e_1, e_2, \psi_1 \dot\vee \psi_2), g, h, \phi) \text{ [:] } \phi}$$

The following tree demonstrates how the goal is deduced.

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

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

$$\dfrac{J_{e_1} \quad \Gamma; \rho_1; \psi_1 \vdash \uparrow^{\mathrm{h}}_1 g \text{ [:] } \phi \quad \Gamma; \rho_1; \psi_2 \vdash \uparrow^{\mathrm{h}}_1 h \text{ [:] } \phi}{\Gamma; \rho_1 \vdash \vee^-(e_1, \uparrow^{\mathrm{h}}_1 g, \uparrow^{\mathrm{h}}_1 h) \text{ [:] } \phi}$$

and $\mathcal{T}_2$ the analogous deduction of $\Gamma; \rho_2 \vdash \vee^-(e_2, \uparrow^{\mathrm{h}}_1 g, \uparrow^{\mathrm{h}}_1 h, \phi)$ [:] ϕ. Now it becomes clear why all variables h_i in, e.g., proof term g for $i \geq 1$ have to be lifted: $\uparrow^{\mathrm{h}}_1 g$. In $\mathcal{T}_1$ the extra assumption ρ_1 is added to the context. The leaves $\Gamma; \rho_1; \psi_1 \vdash \uparrow^{\mathrm{h}}_1 g$ [:] ϕ and $\Gamma; \rho_1; \psi_2 \vdash \uparrow^{\mathrm{h}}_1 h$ [:] ϕ in $\mathcal{T}_1$ are implied by the judgements J_g and J_h respectively, via the weakening lemma (16.2).

PC$\exists\!\to$ Assume $\Gamma \vdash \to^-(\exists^-(d, e, \phi \dot{\to} \psi), f)\ [:]\ \psi$.

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

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 ρ). We have to prove $\Gamma \vdash \exists^-(d, \to^-(e, \uparrow^{\mathrm{h}}(\uparrow^{\mathrm{v}} f)), \psi)\ [:]\ \psi$.

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

Note that $\uparrow(\phi \dot{\to} \psi) = \uparrow\phi \dot{\to} \uparrow\psi$. Thus, all we have to show is that $\uparrow\Gamma; \rho \vdash \uparrow^{\mathrm{h}}(\uparrow^{\mathrm{v}} f)\ [:]\ \uparrow\phi$ follows from $\Gamma \vdash f\ [:]\ \phi$. By Lemma 16.1, we have that $\uparrow\Gamma \vdash \uparrow^{\mathrm{v}} f\ [:]\ \uparrow\phi$. Then our goal follows from the weakening lemma (16.2).

The following theorem, stating that $\rhd$ preserves types, follows directly from Theorem 17.1. The proof proceeds by structural induction on the proposition $d \rhd e$.

Theorem 17.2

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

18 Conclusion

We descibed 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 (Sec. 8) of the deduction system of hypothetical judgements (Sec. 4), 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 intuitionistic first-order logic.

References

[Alt94] Thorsten Altenkirch. *Constructions, inductive types and strong normalisation.* PhD thesis, Laboratory for the Foundations of Computer Science, University of Edinburgh, 1994.

[BB+96] Z. Benaissa, D. Briaud, P. Lescanne and J. Rouyer-Degli, *$\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation.* Functional Programming, 6(5), 1996.

[BW97] Bruno Barras and Benjamin Werner. *Coq in Coq.* **??** 1997.

[BHN00] Marc Bezem, Dimitri Hendriks and Hans de Nivelle. *Automated Proof Construction in Type Theory using Resolution.* In D. McAllester, editor, *Proceedings CADE-17*, volume 1831 of Lecture Notes in Computer Science, pages 148–163. Springer-Verlag, Berlin, 2000.

[Coq99] Bruno Barras et al. *The Coq Proof Assistant Reference Manual*, version 6.3.1, 1999.

[dB72] N.G. de Bruijn. *Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem.* Indagationes Mathematicae volume 34 (5), pages 381–392, 1972.

[Hen98] Dimitri Hendriks. *Clausification of First-Order Formulae, Representation & Correctness in Type Theory.* Master's Thesis, Utrecht University, 1998. URL: `www.phil.uu.nl/~hendriks/thesis.ps.gz`.

[Hue93] Gérard Huet. *Residual theory in lambda calculus, a complete Gallina development.* Rapport de recherche INRIA 2002, 1993.

[MP93] James McKinna and Robert Pollack. *Pure Type Systems Formalized.* In M. Bezem and J.F. Groote, editors, Proceedings 1st Int. Conf. on Typed Lambda Calculi and Applications, TLCA'93, Utrecht, The Netherlands, 16–18 March 1993, volume 664, Springer-Verlag, Berlin, pages 289–305, 1993.

[Per96] Henrik Persson. *Constructive completeness of intuitionistic predicate logic: a formalisation in type theory.* Licentiate thesis, Chalmers University of Technology and University of Götenborg, 1996.

[Pfe96] Frank Pfenning. *The practice of logical frameworks.* In H. Kirchner, editor, Proceedings of the Colloquium on Trees in Algebra and Programming, volume 1059 of Lecture Notes in Computer Science, pages 119–134, Springer-Verlag, 1996.

[Pol96] Jaco van de Pol. *Termination of Higher-order Rewrite Systems.* PhD thesis, Utrecht University, Department of Philosophy, Utrecht, 1996.

[Pra71] D. Prawitz. *Ideas and results in proof theory.* In Jens Erik Fenstad, editor, Proceedings of the Scandinavian Logic Symposium, pages 235-307, Amsterdam, 1971, North-Holland.

[Wer94] Benjamin Werner. *Une Theorie des Constructiones Inductives.* PhD thesis, L'Universite Paris, 1994.