Noname manuscript No.
(will be inserted by the editor)

Formalizing Logical Meta-theory

Semantical Cut-Elimination using Kripke Models for first-order Predicate Logic

Hugo Herbelin · Gyesik Lee

Received: date / Accepted: date

Abstract We propose a new approach to dealing with binding issues when formalizing the metatheory of a logical system with variables such as first-order predicate logic. In our approach, the syntax is represented with *locally traced names*. The style is similar to Pollack-McKinna's locally named approach, in which two sorts of named variables are used. The main difference is that (locally) bound variables occurring in an expression are controlled by the type of that expression. This approach has been adopted to formalize in Coq a Kripke-based semantical cut-elimination of intuitionistic first-order predicate logic.

The five main features of this paper are as follows. First, we show that the roles of constants and free variables can be merged in studying the metatheory of a logical system. Second, there is no need for an extra syntax for well-formed terms and formulae. Third, we emphasize the role of simultaneous substitution and renaming. Fourth, the so-called Exists-Fresh quantification style, the traditional method used to address the binding problems, is revisited. Fifth, the cut-elimination is based on normalization by evaluation (NBE).

During the formalization work, we have attempted to employ common statements for logicians, and we have also retained the simplicity of the programming part.

Keywords Formalization with binders · locally traced names · normalization by evaluation · intuitionistic predicate logic · cut-elimination · Coq

1 Introduction

In formalizing the metatheory of a predicate logic, two sorts of binding are involved: *locally bound* variables are used for representing universal quantification, and *glob-*

Grants or other notes

Hugo Herbelin
INRIA & PPS, Paris Université 7, Paris, France
E-mail: Hugo.Herbelin@inria.fr

Gyesik Lee
Hankyong National University, Anseong-si, Kyonggi-do, Korea
E-mail: gslee@hknu.ac.kr

ally bound variables (that are themselves bound in the right introduction rule of the universal quantification) are used for representing parametric derivations.[1] In the literatures, globally bound variables are also called *free* variables or *parameters*. In this paper, *parameters* are reserved for another use, and we use a more explicit terminology. *Formula-scoped variables* stand for the (locally bound) variables used to represent universal quantification such as x in $\forall x\, P(x)$, whereas *derivation-scoped variables* are (globally bound, i.e., free) variables in parametric derivations such as x in a derivation of the sequent $A(x) \vdash B(x)$.

In traditional (informal) mathematical usage, the same set of variables is used for both formula-scoped and derivation-scoped variables. The main issue in this approach is the possible capture of a derivation-scoped variable by a formula-scoped variable during substitutition of a term in an expression. A typical way of addressing this issue is to ensure that all the derivation-scoped variables are always distinct from the formula-scoped variables. The use of α-conversion makes this possible.

However, during the mechanical development of formal metatheories the representation and manipulation of expressions with variable binding is a tricky issue. The main reason is that α-conversion usually gives rise to a large quantity of extra work. This is an obstacle to formal development in general. To the best of our knowledge, there is no user-friendly nominal representation in Coq, which simulates the traditional practice of using a single set of named variables. (This is because Coq has intentional equality. On the other hand, the nominal representation in Isabelle/HOL uses the extensionality of HOL.)

In this study, we propose a new first-order approach to the formal representation of logical formal systems with variable binding. Previously several widely used technical approaches, each with many variations, have been investigated extensively. We do not try to cover or catalogue these studies here; however, we refer the reader to some papers that include such discussion (cf. [1, 2, 5, 11, 12, 21, 26]).

Our approach is similar to McKinna-Pollack's locally-named representation [20, 21]: Formula-scoped and derivation-scoped variables are represented by two sorts of named variables. The syntactic distinction makes it possible to essentially work with substitution without being concerned with variable capture. A new feature of our approach is that the sets of terms and formulae depend on a list of formula-scoped variables that might be used during the term or formula construction. This is the reason why we call our representation style *representation with locally traced names*. An important consequence is that we can talk about *well-formed* terms and formulae without defining an extra syntax. (Well-formed expressions are those where all formula-scoped variables are actually bound by a binder.)

We remark that there are approaches with two sorts of variables that require no extra syntax for well-formedness. Sato and Pollack [22], e.g., introduced the so-called *internal syntax* where all the expressions are well-formed although two sorts of named variables are used. In fact, no α-conversion is necessary because all the expressions are unique themselves, as this is the case with the approach based on de Bruijn indices.

[1] One may also wonder whether there is no more hidden use of binders in the definition of the proof system based on the fact that some conditions can be imposed on the right implication rule in proofs-as-programs correspondence with λ-abstraction. Indeed, C. Coquand [7] showed that the resulting cut-free proof of a cut-elimination procedure is equivalent to the original proof up to β-like reduction on the proofs seen as λ-terms. However, in this paper, where proofs-as-terms correspondence is not in the program, the right introduction rule of implication can hardly be seen as a form of binding

However, we deliberately renounce to use such approaches because they require special counting mechanisms for deciding binding variables to be used.

To demonstrate the expressiveness and feasibility of our approach, we have formalized in Coq a Kripke-based semantical cut-elimination of intuitionistic first-order predicate logic. The mechanization is carried out in two styles, i.e., with and without derivation-scoped (free) variables. The structures of both mechanizations are nearly identical. This paper is based on the version without derivation-scoped variables. The codes are available online.[2]

Outline of the paper The main features of the paper are summarized in Section 2. In Section 3, we explain derivation-scoped and formula-scoped variables in greater detail. In Section 4, we formally introduce representation with locally traced names. Intuitionistic sequent calculus LJT, Kripke semantics, soundness and completeness are presented in Section 5. In Section 6, we discuss certain issues with respect to our choice of Exists-Fresh quantification style. Section 7 concludes the paper.

2 Main features of the paper

The five main features of this paper are as follows. First, we address a metatheoretic issue on derivation-scoped variables. We investigate the roles of constants and derivation-scoped variables, and we show that derivation-scoped variables can be replaced by fresh constants. In our formalization, only closed formulae are allowed in derivations. This is possible because fresh constants are used instead of derivation-scoped variables.

This is consistent with the fact that free variables are not regarded as a part of a given language. The consequences are manifold: There is no concern regarding variable capture during substitution, substitution is not required for derivation-scoped variables, and a more compact formalization (simpler definitions and shorter proof terms) can be represented.

Second, even if two sorts of variable names are used, an extra syntax for well-formed terms and formulae are not necessary. This significantly reduces the part related to the syntax in formalization.

Third, we emphasize the role of simultaneous substitution and renaming. Using simultaneous substitution, we can establish the relationship between syntax and semantics in a natural way (cf. Universal Completeness Theorem 18).

Further, they are essential to show that the Exists-Fresh style is adequate for dealing with quantification.[3] And the equivalence of three well-known different quantification styles (the Exists-Fresh, Cofinite, and All-Fresh styles) can be easily proved. Although the equivalence is already known, our proof reveals certain new aspects of simultaneous substitution and renaming. In fact, it is not necessary to have any kind of strengthened induction principles such as the well-founded induction on the length of an expression or on the length of a proof.

Fourth, the so-called Exists-Fresh quantification style, the traditional method of dealing with the binders, is used in the formalization:

$$\frac{\Gamma \vdash A[x\backslash c] \quad \text{for some } c \notin OC(A, \Gamma)}{\Gamma \vdash \forall x\, A} \ (\textit{Exists-Fresh})$$

[2] Cf. Coq files with names of the form `without_FreeVar_*.v` in our library at `http://ropas.snu.ac.kr/~gslee/Publi/Kripke.zip`

[3] See also the discussion in Section 4 of Aydemir et al. [2].

Here, $A[x\backslash c]$ denotes the formula A, where the constant c is substituted for x, and $OC(\Delta)$ denotes the set of constants occurring in the context Δ. (Note that we use fresh constants instead of fresh derivation-scoped variables.)

We chose the Exists-Fresh style because it closely resembles the pen-and-paper style. In spite of its unpopularity, we believe that it is the best approach when one wants to follow the usual style of mathematical logic. Here, we show how the difficulties in using the Exists-Fresh style in a formalization work can be addressed.

Fifth, the cut-elimination is based on normalization of evaluation for intuitionistic first-order predicate logic: A composition of completeness and soundness leads to cut-elimination. Refer to Berger et al. [3] and Berger and Schwichtenberg [4] for more details regarding normalization by evaluation.

Because all the proofs are constructive, our work can be seen as an extension of Coquand's work [7,8], where first-order propositional logic is the main object.

3 Variables, constants, and α-conversion

Derivation-scoped and formula-scoped variables When we look at the following informal right introduction rule of $\forall$-quantification, we can make some observations:

$$\frac{\Gamma \vdash A(y) \qquad y \text{ fresh in } \Gamma, A}{\Gamma \vdash \forall x A(x)}$$

First, the $\forall$-quantification is part of the domain of discourse because it is generally considered up to α-equivalence: The actual name of x in $\forall x A(x)$ is not relevant.

Second, the $\forall$-quantifiers are instantiated by terms that do not contain formula-scoped variables. Instantiation of quantifiers by terms occurs in the left introduction rule of the $\forall$-quantification and in the definition of the semantics of universally quantified formulae.

Third, there is no need to consider instantiation of derivation-scoped variables in the definition of the deduction system.

Fourth, if derivability is a predicate and not a part of the domain of the discourse, there is no need to consider derivation up to the actual name of the derivation-scoped variables.

However, the need for α-conversion over derivation-scoped variables and instantiation of derivation-scoped variables will arise if derivability is lifted as the object of the domain of discourse as in Coquand [7,8]. She showed that the cut-free derivation obtained by semantic cut-elimination actually implements β-normalization and η-expansion over derivations.

Derivation-scoped variables as constants The intended meaning of a derivation in which derivation-scoped variables occur (e.g., in $A(x) \vdash B(x)$) is the collection of all derivations obtained by instantiating the variables by arbitrary terms. However, derivation-scoped variables are not instantiated by any rule. Further, as we will see in Lemma 16 and Lemma 23, we can replace a fresh constant with arbitrary terms when it does not occur in any assumptions.

All these facts suggest that derivation-scoped variables can be regarded as constants even though this might appear counter-intuitive: A constant is supposed to represent a given object, and not a collection. In fact, special attention is required in establishing relations between syntax and semantics (cf. the soundness proof of Theorem 14).

An important consequence of considering derivation-scoped variables as constants at the syntactic level is that substitution is needed only for formula-scoped variables. The language itself needs to contain infinitely many constants. Note that every language can be conservatively extended to a language with infinitely many constants.

It is noteworthy that even if we formally introduce derivation-scoped variables, the substitution for these variables do not play any role (cf. our work with derivation-scoped variables, i.e., the Coq files with names of the form `with_FreeVar_*.v`.).

On the purpose of α-conversion In our work, terms that differ only by the names of formula-scoped variables are not considered to be equivalent. This may be surprising as it departs from the common usage of reasoning modulo α-conversion. However, in practice, it is harmless because for any derivation that would consider some formulae modulo α-conversion, there is another that differs from the original only by the names of formula-scoped variables, and that does not need α-conversion.

Let $\vdash$ be a *first-order* proof system (such as the one in Figure 4) and $\vdash_{\overset{\alpha}{\equiv}}$ its extension with the rule

$$\frac{\Gamma \vdash A \quad \Gamma \overset{\alpha}{\equiv} \Gamma' \quad A \overset{\alpha}{\equiv} A'}{\Gamma' \vdash A'}$$

where $A \overset{\alpha}{\equiv} A'$ is the α-conversion on formulae and $\Gamma \overset{\alpha}{\equiv} \Gamma'$ is its canonical extension to contexts. The following postponement result justifies the uselessness of α-conversion:

Lemma 1 *$\Gamma \vdash_{\overset{\alpha}{\equiv}} A$ iff there exists $\Gamma' \overset{\alpha}{\equiv} \Gamma$ and $A' \overset{\alpha}{\equiv} A$ such that $\Gamma' \vdash A'$*

The proof is by induction; we have to ensure that α-conversion commutes with every rule of the logic. The rules that instantiate a binder require special attention; let us assume that the last rule of the derivation has one of the following forms:

$$\frac{\Gamma \vdash_{\overset{\alpha}{\equiv}} A(t)}{\Gamma \vdash_{\overset{\alpha}{\equiv}} \forall x\, A(x)} \qquad \frac{\Gamma, A(t) \vdash_{\overset{\alpha}{\equiv}} B}{\Gamma, \forall x\, A(x) \vdash_{\overset{\alpha}{\equiv}} B}$$

By induction there is $A'(t) \overset{\alpha}{\equiv} A(t)$, $\Gamma' \overset{\alpha}{\equiv} \Gamma$, and $B' \overset{\alpha}{\equiv} B$ such that $\Gamma' \vdash A'(t)$ or $\Gamma', A'(t) \vdash B'$ hold. Subsequently, we merely have to select a fresh y not used as a binder in A' and build $\forall y\, A'(y)$ which by construction satisfies $\forall y\, A'(y) \overset{\alpha}{\equiv} \forall x\, A(x)$. □

4 Representation with locally traced names

The usage of two type of variable names was first implemented by McKinna and Pollack [20,21] to formalize the Pure Type System (PTS) meta-theory, following the suggestion by Coquand [9]. However, there are certain limitations; not all syntactic expressions are meaningful because formula-scoped variables could occur unbound. This is also the case, even though derivation-scoped variables are replaced by constants. Therefore, it is usually required to provide an extra syntax for the definition of well-formed expressions. In McKinna and Pollack [20,21], well-formed expressions are called *variable-closed* while they are called *locally-closed* in Aydemir et al. [2]. We prefer well-formedness notion because in earlier works on mathematical logic (e.g., by Church [6]), well-formed formulas were used to denote the strings that followed the formation rules of *correct* formulas.

Pseudo-terms:

$$\frac{x \in \mathtt{nat} \quad (h : x \in m)}{\mathtt{Bvar}\, x\, h \in \mathtt{pterm}\, m} \qquad \frac{c \in \mathtt{nat}}{\mathtt{Cst}\, c \in \mathtt{pterm}\, m} \qquad \frac{f \in \mathtt{function} \quad t_1, t_2 \in \mathtt{pterm}\, m}{\mathtt{App}\, f\, t_1\, t_2 \in \mathtt{pterm}\, m}$$

where $(h : x \in m)$ denotes that h is the proof that x is contained in the list m.

Pseudo-formulae:

$$\frac{P \in \mathtt{predicate} \quad t \in \mathtt{pterm}\, m}{\mathtt{Atom}\,(p, t) \in \mathtt{pformula}\, m} \qquad \frac{A \in \mathtt{pformula}\, m \quad B \in \mathtt{pformula}\, m}{\mathtt{Imply}\, A\, B \in \mathtt{pformula}\, m}$$

$$\frac{x \in \mathtt{nat} \quad A \in \mathtt{pformula}\,(x :: m)}{\mathtt{Forall}\, x\, A \in \mathtt{pformula}\, m}$$

Notations: $P\, t = \mathtt{Atom}(P, t), \qquad A \to B = \mathtt{Imply}\, A\, B, \qquad \forall x\, A = \mathtt{Forall}\, x\, A.$

Contexts: $\mathtt{context} = \mathtt{list}\ \mathtt{formula} = \mathtt{list}\ (\mathtt{formula}\ nil)$

Occurrence of constants:

$$\begin{aligned} \mathtt{OC}(\mathtt{Bvar}\, x\, h) &= \varnothing & \mathtt{OC}(P\, t) &= \mathtt{OC}(t) \\ \mathtt{OC}(\mathtt{Cst}\, c) &= \{c\} & \mathtt{OC}(A \to B) &= \mathtt{OC}(A) \cup \mathtt{OC}(B) \\ \mathtt{OC}(\mathtt{App}\, f\, t_1\, t_2) &= \mathtt{OC}(t_1) \cup \mathtt{OC}(t_2) & \mathtt{OC}(\forall x\, A) &= \mathtt{OC}(A) \end{aligned}$$

Occurrence of formula-scoped variables:

$$\begin{aligned} \mathtt{BV}(\mathtt{Bvar}\, x\, h) &= \{x\} & \mathtt{BV}(P\, t) &= \mathtt{BV}(t) \\ \mathtt{BV}(\mathtt{Cst}\, c) &= \varnothing & \mathtt{BV}(A \to B) &= \mathtt{BV}(A) \cup \mathtt{BV}(B) \\ \mathtt{BV}(\mathtt{App}\, f\, t_1\, t_2) &= \mathtt{BV}(t_1) \cup \mathtt{BV}(t_2) & \mathtt{BV}(\forall x\, A) &= \mathtt{BV}(A) \backslash \{x\} \end{aligned}$$

Fig. 1 Pseudo-terms and -formulae without free variables

Here, we introduce a new first-order approach, where no extra syntax is necessary for well-formed expressions, even though formula-scoped and derivation-scoped variables are syntactically distinguished. The explanation will be provided by demonstrating how to formalize intuitionistic first-order predicate logic with respect to Kripke semantics. For the presentation of predicate logic, we adopt sequent calculus to represent proofs. The advantage of such an approach is that it has an easy-to-define notion of the normal form (it is merely the absence of the cut rule). A disadvantage is that it is less *natural* than the so-called natural deduction; however, such a structure has already been used by Coquand [7] and we found it interesting to try an alternative approach.

The language we consider contains $\to$ and $\forall$ as the sole connectives as well as countably many constants. We use natural numbers to denote both formula-scoped variables and constants. `Bvar` stands for the denotation of formula-scoped variables while `Cst` represents constants. We let c, d, c_i, d_i vary over constants while x, y, x_i, y_i vary over formula-scoped variables. For simplicity, we assume two denumerable and decidable sets: `predicate` of unary predicates and `function` of binary functions. Note that a set A is *decidable* if, constructively, $\forall a, b \in A\,(a = b \vee a \neq b)$, otherwise said, if there exists a decision function f from $A \times A$ such that $a = b \leftrightarrow f(a, b) = 0$. The symbols f, g, f_i, g_i (resp. P, Q, P_i, Q_i) denote function (resp. predicate) symbols.

Remark 2 *For the formalization, we use (finite) lists to denote finite sets of constants, variables, or formulae. For our purpose, it is sufficient to define sublist in a settheoretic manner: A list ℓ is a sublist of another list k if ℓ is a subset of k when they are regarded*

as sets. (The base library for sublist is called `sublist.v`*. It contains 1 definition and 16 very simple lemmata.)*

However, in this paper, we also use the usual set notations for better readability: the singleton $\{x\}$ *for* $x :: nil$*, the set union operator* $\cup$ *for the concatenation of two lists,* $\ell \backslash m$ *for the set-theoretic abstraction,* $\ell \backslash \{x\}$ *or* ℓ^{-x} *for the removal of* x *from the list* ℓ*, and the element relation* $\in$ *for the* being-in-a-list *relation.*

Terms and formulae as inductive families Given a list of formula-scoped variables $m \in \texttt{list nat}$, $\texttt{pterm}\, m$ and $\texttt{pformula}\, m$ denote the set of pseudo-terms and pseudo-formulae, respectively (see Figure 1). Intuitively, the parameter m collects the formula-scoped variables that possibly occur unbound. We use the notation "*pseudo-*" because some formula-scoped variables can occur unbound while they are supposed to be bound. (Note that our approach follows also the usage, common in the theory of lambda calculus, to have a notation for the set of terms over some set of variables.)

The side condition $(h : x \in m)$ in the definition of $\texttt{Bvar}\, x\, h \in \texttt{pterm}\, m$ is a crucial feature of the entire formalization. In this manner, we control the information on variables used in pseudo-terms or -formulae:

Lemma 3 *Let* $e \in \texttt{term}\, m$ *or* $e \in \texttt{pformula}\, m$*. Then,* $\texttt{BV}(e) \subseteq m$*. In particular,* $\texttt{BV}(e)$ *is an empty list when* e *is a well-formed term or a formula.*

In particular, this means that the well-formed terms (resp. formulae) are represented by the elements of $\texttt{pterm}\, nil$ (resp. $\texttt{pformula}\, nil$):

$$\texttt{term} := \texttt{pterm}\, nil \quad \text{and} \quad \texttt{formula} = \texttt{pformula}\, nil$$

In addition, there are meta-level reasons as to why the compact handling of well-formed expressions is important. First, only well-formed terms and formulae have a meaning and correspond to ordinary terms and formulae in the traditional nominal representation. Another point is that some useful properties hold only for well-formed terms and formulae. For example, deduction rules are intended to be applied only for well-formed formulae (see Figure 4).

Syntactic decidability Because we work with sequent calculus, it is neccesary to check whether a pseudo-formula occur in a context, which is a list of (well-defined) formulae (see Figure 4). For this, we need to decide the syntactic equality of two expressions.

The syntactic decidability suggests another important point of our approach: α-equivalence of formulae is the equality of the underlying skeleton where names have been dropped while proofs "h" have been kept. In some sense, this is very similar to what happens in Coq where names are kept internally for printing purposes while de Bruijn indices are used for reference purposes. With our definition, the important part is the proof "h", which is a canonical index (some "n^{th}" from the list m, as explained in Lemma 3). The names in this case are inessential, with m specifying the order in which the names are numbered. Compared to the standard locally named representation that has no control of the exposed bound, in our case, we superimpose de Bruijn indices and names so that α-equivalence (implicit in Theorem 5 below) is easily checked.

Let $t \in \mathtt{pterm}\, m$, $\mathtt{BV}(t) \subseteq \ell$. Then $\mathtt{tlift}^{m,\ell} : \mathtt{pterm}\, m \to \mathtt{pterm}\, \ell$ is recursively defined: ($\mathtt{tlift}$ denotes $\mathtt{tlift}^{m,\ell}$.)

$$\begin{aligned} \mathtt{tlift}(\mathtt{Bvar}\, x\, h) &= \mathtt{Bvar}\, x\, h' \\ \mathtt{tlift}(\mathtt{Cst}\, c) &= \mathtt{Cst}\, c \\ \mathtt{tlift}(\mathtt{App}\, f\, t_1\, t_2) &= \mathtt{App}\, (\mathtt{tlift}(t_1))\, (\mathtt{tlift}(t_2)) \end{aligned} \tag{1}$$

where h' is a proof of $x \in \ell$, which can be obtained from the assumption $\mathtt{BV}(t) \subseteq \ell$.

Let $A \in \mathtt{pformula}\, m$, $\mathtt{BV}(A) \subseteq \ell$. Then, $\mathtt{flift}^{m,\ell} : \mathtt{pformula}\, m \to \mathtt{pformula}\, \ell$ is recursively defined: ($\mathtt{flift}$ denotes $\mathtt{flift}^{m,\ell}$.)

$$\begin{aligned} \mathtt{flift}(P\, t) &= P\, (\mathtt{tlift}(t)) \\ \mathtt{flift}(A \to B) &= \mathtt{flift}(A) \to \mathtt{flift}(B) \\ \mathtt{flift}(\forall x\, C) &= \forall x\, (\mathtt{flift}(C)) \end{aligned}$$

where $\mathtt{flift}(C) = \mathtt{flift}^{x::m,x::\ell}(C)$ depends on the proof of $\mathtt{BV}(C) \subseteq x :: \ell$ which trivially follows from $\mathtt{BV}(A) \subseteq \ell$.

Fig. 2 Lifting

To decide whether two pseudo-terms are syntactically equal, the being-in-a-list condition should be decidable. Therefore, we use the following definition that is equivalent[4] to the standard one from the Coq library for lists: Given a decidable set A and $m \in \mathtt{list}\, A$,

$$x \in y :: m \quad \text{iff} \quad \text{if } x = y \text{ then } \mathtt{True} \text{ else } x \in m.$$

This definition enables us to have the proof unicity of being-in-a-list relation: Given a parameter, the proof part in the definition of a formula-scoped variable is uniquely defined.

Lemma 4 (Proof unicity of being-in-a-list) *Let A be a decidable set, $a \in A$, and $m \in \mathtt{list}\, A$. Then*

$$\forall (p, q : a \in m)\, (p = q)\,.$$

Now, the syntactic decidability follows easily.

Theorem 5 (Syntactic decidability) *Let m be a finite list of natural numbers.*

1. $\forall (t, s : \mathtt{pterm}\, m)\, (t = s \ \vee \ t \neq s)$.
2. $\forall (A, B : \mathtt{pformula}\, m)\, (A = B \ \vee \ A \neq B)$.

Parameter lifting Even if we have the proof unicity of being-in-a-list, a pseudo-term or a -formula belongs to infinitely many classes: a term t from $\mathtt{pterm}\, m$ belongs also to $\mathtt{pterm}\, k$ for all parameter k including m as a sublist.

However, note that for any pseudo-term $t \in \mathtt{pterm}\, m$, the set $\ell = \mathtt{BV}(t)$ is the canonical and smallest parameter such that $t \in \mathtt{pterm}\, \ell$. (Note that we are considering lists as finite sets.) We use this property to connect the two classes $\mathtt{pterm}\, m$ and $\mathtt{pterm}\, k$ by a *homomorphic lifting* function. The lifting function is homomorphic in the sense that it preserves syntactic and semantic properties such as syntactic decidability, substitution, and validity in a Kripke model. (See Lemma 7, Lemma 9, and Theorem

[4] This equivalence enabled us to freely access all the existing standard libraries for lists.

Recursive definition of $t\,[\,\eta \triangleright \ell\,] \in \mathtt{pterm}\,\ell$:

$$(\mathtt{Bvar}\,y\,h)\,[\,\eta \triangleright \ell\,] = \begin{cases} \mathtt{Bvar}\,y\,h' & \text{if } y \in \ell \\ \mathtt{tlift}\,u_j\,h_j & \text{if } y \notin \ell \text{ and } j = \min\{i : y = x_i\} \\ \mathtt{Cst}\,0 & \text{otherwise} \end{cases} \tag{2}$$

$$(\mathtt{Cst}\,c)\,[\,\eta \triangleright \ell\,] = \mathtt{Cst}\,c$$

$$(\mathtt{App}\,f\,t_1\,t_2)\,[\,\eta \triangleright \ell\,] = \mathtt{App}\,f\,(t_1\,[\,\eta \triangleright \ell\,])\,(t_2\,[\,\eta \triangleright \ell\,])$$

where

- h' is the proof of $y \in \ell$ given by the assumption,
- h_j is a proof-term witnessing $\mathtt{BV}(u_j) = nil \subseteq \ell$.

Recursive definition of $A\,[\,\eta \triangleright \ell\,] \in \mathtt{pformula}\,\ell$:

$$(P\,t)\,[\,\eta \triangleright \ell\,] = P\,(t\,[\,\eta \triangleright \ell\,])$$

$$(A \to B)\,[\,\eta \triangleright \ell\,] = A\,[\,\eta \triangleright \ell\,] \to B\,[\,\eta \triangleright \ell\,]$$

$$(\forall x\,B)\,[\,\eta \triangleright \ell\,] = \forall x\,(B\,[\,\eta \triangleright x :: \ell\,]) \tag{3}$$

Fig. 3 Simultaneous substitution with destination

11.) We remark that parameter lifting is necessary to deal with substitution. (see Figure 3).

The lifting function presented in Figure 2 is based on the fact that the being-in-a-list part is inessential so long as the parameter contains all the formula-scoped variables needed for the construction of an expression: Note that in the definition of $\mathtt{tlift}^{m,\ell}$ (resp. $\mathtt{flift}^{m,\ell}$), we demand $\mathtt{BV}(t) \subseteq \ell$ (or $\mathtt{BV}(A) \subseteq \ell$). Further, it is noteworthy that $\mathtt{tlift}^{m,\ell}(t)$ does not change anything in t, but in the being-in-a-list part, and that the choice of h' in (1) is not important because of the proof unicity, Theorem 4. In addition, this indicates that repeated application of lifting is equivalent to a single application and that the syntactic equality between two expressions is preserved.

Lemma 6 (Idempotence) *Assume* $e \in \mathtt{pterm}\,m$ *or* $e \in \mathtt{pformula}\,m$ *and* $m \subseteq \ell \subseteq k$. *Then,*

$$\mathtt{lift}^{\ell,k}(\mathtt{lift}^{m,\ell}(e)) = \mathtt{lift}^{m,k}(e)$$

where `lift` *denotes* `tlift` *or* `flift` *depending on* e.

The following lemma says that lifting functions preserve equality.

Lemma 7 (Equality preservation) *Assume* $e, e' \in \mathtt{pterm}\,m$ (*or* $e, e' \in \mathtt{pformula}\,m$) *and* $\mathtt{BV}(e), \mathtt{BV}(e') \subseteq \ell$. *Then,* $\mathtt{lift}^{m,\ell}(e) = \mathtt{lift}^{m,\ell}(e')$ *denotes* $e = e'$. *Here,* `lift` *denotes* `tlift` *or* `flift` *depending on* e, e'.

The overhead for dealing with lifting is very small. In addition to the two definitions for `tlift` and `flift`, we needed only 14 lemmata, which are all proved in several lines. (The base library for lifting is called `without_FreeVar_lifting.v` in our library.)

Substitution with destination The definition of a substitution requires special consideration because of two reasons.

First, the relationship between syntax and semantics such as soundness and completeness should be realized in a natural way. For this purpose, we use simultaneous

substitution instead of typical single substitution. Universal Completeness Theorem 18, e.g., shows a natural correspondence between semantics and syntax by means of simultaneous substutition. In Section 6, we will also see that simultaneous renaming of constants, which can be considered as a special form of simultaneous substitutions, plays an important role. Refer to Stoughton [23] and Sato and Pollack [22] for further details regarding the usefulness of simultaneous substitution.

The simultaneous substitution of finitely many terms for formula-scoped variables in a pseudo-term or a -formula is defined by the structural recursion shown in Figure 3. Only well-formed terms are substituted because substituting pseudo-terms could cause variable capture. In this way, we avoided α-conversion during the entire formalization.

Second, because of the parameter part, it is not clear to which class the resulting expression of a subtitution should belong. Thus, we state cleary in the definition of substitution the type of the resulting expression.

Given a pseudo-term or a -formula e, a finite list of natural numbers ℓ, and an association $\eta = (x_1, u_1), ..., (x_n, u_n)$, where $x_i \in \mathtt{nat}$ and $u_i \in \mathtt{term}$, the simultaneous substitution has the following form $e\,[\,\eta \triangleright \ell\,]$. The single substitution $e\,[\,x \setminus u\,]_\ell := e\,[\,(x,u) \triangleright \ell\,]$ is then a special case. Furthermore, we write $e\,[\,x \setminus u\,]$ when $\ell = nil$ for better readability.

Intuitively, ℓ is the list of formula-scoped variables for which no substitutions are allowed: The type of the resulting expression is independent of the type of e. By extending ℓ by x in the abstraction case (3), we enforce the rule that any substitution for x is forbidden. Note that we extend ℓ instead of changing η, e.g., by deleting any occurrence of (x, u) from $(x_1, u_1), ..., (x_n, u_n)$. Further, we abandon all the insignificant variables, variables occurring neither in ℓ nor in $\bar{x}$ (cf. (2)). Consequently, we get $t\,[\,\eta \triangleright \ell\,] \in \mathtt{pterm}\,\ell$ and $A\,[\,\eta \triangleright \ell\,] \in \mathtt{pformula}\,\ell$.

Therefore, the type of $e\,[\,\eta \triangleright \ell\,]$ depends only on ℓ. In particular, we have $t\,[\,\eta \triangleright nil\,] \in \mathtt{term}$ and $A\,[\,\eta \triangleright nil\,] \in \mathtt{formula}$, which enables us to give more intuitive definitions and proofs (cf. the rule $(\forall_L)$ in Figure 4). For example, the left introduction rule of universal quantification can be formalized in the following form

$$\frac{\Gamma \mid A\,[\,x \setminus t\,]_{nil} \vdash C}{\Gamma \mid \forall x\, A \vdash C}$$

without referring to the well-formedness of $A\,[\,x \setminus t\,]_{nil}$, see Figure 4.

In order to demonstrate that the substitution behaves as expected, we state two lemmata:

Lemma 8 (Substitution Lemma) *Let* $e \in \mathtt{pterm}\,m$ *or* $e \in \mathtt{pformula}\,m$, $u \in \mathtt{term}$, *an association* η, *and* $\ell \in \mathtt{list}\,\mathtt{nat}$. *Then*

$$(e\,[\,\eta \triangleright y :: \ell\,])\,[\,y \setminus u\,]_\ell = e\,[\,(y,u) :: \eta \triangleright \ell\,]\,.$$

Lemma 9 (Lifting and substitution) *Let* $e \in \mathtt{pterm}\,m$ *or* $e \in \mathtt{pformula}\,m$, $\mathtt{BV}(e) \subseteq k$. *Then*

$$(\mathtt{lift}^{m,k}(e))\,[\,\eta \triangleright \ell\,] = e\,[\,\eta \setminus \ell\,]\,,$$

where `lift` *is* `tlift` *or* `flift` *depending on* e. *That is, lifting has no impact on substitution.*

$$\frac{}{\Gamma \mid A \vdash A}\ (Ax) \qquad \frac{\Gamma \mid A \vdash C \quad A \in \Gamma}{\Gamma \vdash C}\ (Contr)$$

$$\frac{\Gamma \vdash A \quad \Gamma \mid B \vdash C}{\Gamma \mid A \to B \vdash C}\ (\to_L) \qquad \frac{A :: \Gamma \vdash B}{\Gamma \vdash A \to B}\ (\to_R)$$

$$\frac{\Gamma \mid A\,[x \setminus t] \vdash C}{\Gamma \mid \forall x\, A \vdash C}\ (\forall_L) \qquad \frac{\Gamma \vdash A\,[x \setminus c] \quad \text{for some } c \notin \mathtt{OC}(A, \Gamma)}{\Gamma \vdash \forall x\, A}\ (\textit{Exists-Fresh-}\forall_R)$$

Fig. 4 Cut-free LJT

5 Intuitionistic sequent calculus LJT and Kripke semantics

Having defined the basic syntax, we provide in this section the deduction rules and a Kripke semantics for the Gentzen-style sequent calculus LJT. We will establish that LJT is sound and complete with respect to the Kripke semantics.

5.1 Intuitionistic sequent calculus LJT

For the presentation of predicate logic, we adopt sequent calculus to represent proofs. The advantage of such an approach is that it has an easy-to-define notion of normal form (it is merely the absence of the cut rule). A disadvantage is that it is less natural than the so-called natural deduction. However, such a structure has already been used in Coquand [7] and we found interesting to try an alternative approach.

The Gentzen-style sequent calculus LJT presented in Figure 4 is obtained from the intuitionistic sequent calculus LJ by restricting the use of the left introduction rules of the implication and the universal quantification. A sequent has one of the forms $\Gamma \mid A \vdash C$ or $\Gamma \vdash C$, where A, C are well-formed formulae and the context $\Gamma = A_1, ..., A_n$ is a list of well-formed formulae. The right side of "|" in the antecedent is called *stoup*, and it contains the principal formula (Hauptformel) of the corresponding rule.

Herbelin [13,14] showed that one can attain a one-to-one correspondence between cut-free proofs in LJT and normal λ-terms. To be more precise, cut-elimination matches normalization in the $\overline{\lambda}$-calculus, which is a suitable variant of λ-calculus for the sequent calculus structure. This implies that LJT is a Curry-Howard-de Bruijn-style proof system.

Remark 10 *The Curry-Howard-de Bruijn approach requires the ability to distinguish between the different occurrences of a given formula in the context Γ of a sequent $\Gamma \vdash A$. There are two canonical ways to achieve this: one either considers contexts as sets of* named *formulae, where the names are used to distinguish between the different occurrences of the same formula, or one considers contexts as list (i.e., ordered sets) of formulae, in which case the underlying order provides a way to distinguish between different occurrences of the same formula.*

On the other hand, considering contexts as sets precludes the correspondence with λ-calculus as, for instance, there would be only one proof of $A \to A \to A$ while there are two λ-terms of type $A \to A \to A$. Strictly speaking, considering contexts as multisets does not help since there is no way to distinguish between two instances of the

same formula in a multiset. If the multiset is equipped with a convention to distinguish between the different occurrences of the same formula (e.g., Troelstra and Van Dalen's crude discharge convention [25, Ch. 1]), this amounts to giving names to formulae (see Geuvers [10] for a discussion).

In the current work, we use lists of formulae to represent contexts. However, they are regarded as finite sets. Note that derivability is a predicate and not a part of the domain of the discourse, i.e., Curry-Howard-de Bruijn correspondence itself is not on the program.

We emphasize that all the formulae occurring in derivations are well-formed formulae and that the deduction rules can be represented exactly as in Figure 4. This is possible because we can primarily focus on $\texttt{formula} = \texttt{pformula}\, nil$. Note that typing rules are usually defined for arbitrary pseudo-terms, and then people show that only well-formed expressions are involved in typing rules.

If we had followed, e.g., the locally-named style of McKinna and Pollack [20,21], we should have defined the deduction rules with pseudo-formulae, implicitly thinking of well-formed formulae. The rule (Ax), e.g., would need a side condition that all formulae involved are well-formed:

$$\frac{Ok(A :: \Gamma)}{\Gamma \mid A \vdash A}$$

where $Ok(\Gamma)$ denotes that all formulae in Γ are well-formed formulae. Subsequently, we could prove the following: if $\Gamma \vdash A$ or $\Gamma \mid a \vdash C$ then $Ok(A :: C :: \Gamma)$. As we have already mentioned, this approach requires extra syntax and properties about well-formed formulae and contexts (cf. McKinna and Pollack [20,21] and Aydemir et al. [2]).

5.2 Kripke semantics

Kripke semantics was created in the late 1950s and early 1960s by Saul Kripke [17,18]. It was first made for modal logic, and later adapted to intuitionistic logic and other non-classical or classical systems (cf. Troelstra and van Dalen [24] and Ilik et al. [16]). Here, we use the conventional Kripke model adopted by Troelstra and van Dalen [24] for the semantics of LJT.

A Kripke model $\mathcal{K} = (\mathcal{W}, \leq, \Vdash, \mathcal{D}, V)$ is a tuple of a partially-ordered set $\mathcal{W}$ of *worlds*, a domain $\mathcal{D}$, interpretations of constant and function symbols into the domain, and a binary relation between worlds and atomic sentences in the extended language with the new constant symbols for each domain element (cf. Figure 5).

Note that the interpretation of pseudo-terms is made total by ignoring insignificant formula-scoped variables. The being-in-a-list part of a term is simply ignored during the interpretation. Consequently, the lifting of formula-scoped variables has no impact on the Kripke semantics.

Theorem 11 (Lifting-irrelevance) *Given a pseudo-term* $t \in \texttt{pterm}\, m$ *or a pseudo-formula* $A \in \texttt{pformula}\, m$*, we have*

$$t[\eta] = \texttt{tlift}(t)[\eta] \quad \textit{and} \quad (w \Vdash A[\eta] \ \textit{iff}\ w \Vdash \texttt{flift}(A)[\eta]).$$

The monotonicity of the forcing relation with respect to the worlds relation $\leq$ can be proved by a simple structural induction:

Kripke models: $\mathcal{K} = (\mathcal{W}, \leq, \Vdash, \mathcal{D}, V)$, where $(\mathcal{W}, \leq)$ is a partially ordered set, $\mathcal{D}$ is the domain of $\mathcal{K}$, V is a function such that

1. $V(c) \in D$ for all $c \in \mathtt{nat}$,
2. $V(f) : \mathcal{D} \to \mathcal{D} \to \mathcal{D}$ for all $f \in \mathtt{function}$,

and $\Vdash$ is a relation between $\mathcal{W}$ and the set of atomic sentences in the language extended with constant symbols for each element of $\mathcal{D}$ such that

$$(w \leq w' \quad \text{and} \quad w \Vdash P\,d) \quad \Rightarrow \quad w' \Vdash P\,d$$

where $w, w' \in \mathcal{W}$, $P \in \mathtt{predicate}$, and $d \in \mathcal{D}$.

Interpretation of pseudo-terms: Let $\eta \in \mathtt{list}\,(\mathtt{nat} * \mathcal{D})$

$$\begin{aligned} (\mathtt{Bvar}\,x\,h)[\eta] &= \begin{cases} \eta(x) & \text{if } x \in \mathsf{dom}(\eta) \\ V(0) & \text{otherwise} \end{cases} \\ (\mathtt{Cst}\,c)[\eta] &= V(c) \\ (f\,t_1\,t_2)[\eta] &= V(f)(t_1[\eta], t_2[\eta]) \end{aligned} \tag{4}$$

Here $\eta(x) = d$ if (x, d) is the first occurence in η from left.

Forcing: The relation $\Vdash$ is inductively extended to general sentences in the extended language.

$$\begin{aligned} w \Vdash (P\,t)[\eta] &\text{ iff } w \Vdash P\,(t[\eta]) \\ w \Vdash (A \to B)[\eta] &\text{ iff for all } w' \geq w,\ w' \Vdash A[\eta] \text{ implies } w' \Vdash B[\eta] \\ w \Vdash (\forall x\,A)[\eta] &\text{ iff for all } d \in \mathcal{D},\ w \Vdash A[(x, d) :: \eta] \end{aligned} \tag{5}$$

$$w \Vdash \Gamma \text{ iff } w \Vdash A[nil] \text{ for all } A \in \Gamma$$

We sometimes write $\Vdash_{\mathcal{K}}$ when necessary.

Fig. 5 Kripke semantics

Lemma 12 (Monotonicity) *Given a pseudo-formula $A \in \mathtt{pformula}\,m$, if $w \Vdash A[\eta]$ and $w \leq w'$ hold, so does $w' \Vdash A[\eta]$.*

Remark 13 *The standard Kripke semantics uses cumulative domains $\mathcal{D}(w), w \in \mathcal{W}$ instead of a fixed domain $\mathcal{D}$ such that $\mathcal{D}(w) \subseteq \mathcal{D}(w')$ when $w \leq w'$. Then, the universal quantification case* (5) *should have the following form:*

$$w \Vdash (\forall x\,A(x))[\eta] \textit{ iff, for all } w' \geq w \textit{ and } d \in D(w),\ w' \Vdash A\,[(x, d) :: \eta].$$

In our case, where $\to$ and $\forall$ are the only logical symbols, two styles are "functionally equivalent" in the sense that soundness and completeness hold in both cases (cf. Herbelin and Lee [15]).

5.3 Soundness

The soundness of LJT with respect to the Kripke semantics is relatively simple.

Theorem 14 (Soundness) *Given a Kripke model $\mathcal{K} = (\mathcal{W}, \leq, \Vdash_{\mathcal{K}}, \mathcal{D}, V)$ and an assoction $\eta \in \mathtt{list}\,(\mathtt{nat} * \mathcal{D})$, we have*

1. *If $\Gamma \vdash C$ and $w \Vdash_{\mathcal{K}} \Gamma$ hold, then $w \Vdash_{\mathcal{K}} C[\eta]$.*

2. If $\Gamma \mid A \vdash C$, $w \Vdash_{\mathcal{K}} \Gamma$, $w \Vdash_{\mathcal{K}} A[\eta]$ *hold, then* $w \Vdash_{\mathcal{K}} C[\eta]$.

If we had included derivation-scoped variables (free variables), the soundness proof of LJT would be a simple simultaneous induction on the derivation, cf. Herbelin and Lee [15]. However, because we use constants instead, the (*Exists-Fresh*-$\forall_R$) rule requires more attention.

Let us assume that $\Gamma \vdash \forall x\, A$ follows from $\Gamma \vdash A\,[\,x \setminus c\,]$ for a constant $c \notin \mathtt{OC}(A, \Gamma)$. Given an arbitrary $d \in \mathcal{D}$, we have to show that

$$w \Vdash_{\mathcal{K}} A[(x, d) :: \eta] \tag{6}$$

holds using the induction hypothesis that $\omega \Vdash_{\mathcal{K}} \Gamma$ and $w \Vdash_{\mathcal{K}} A[(x, d_0) :: \eta]$ where $d_0 := V(c)$. At this point, the premise of (*Exists-Fresh*-$\forall_R$) seems to provide too weak an induction hypothesis because a constant is associated with a *fixed* value, while the interpretation of the universal quantification involves all possible values from the domain.

The solution lies in the fact that fresh constants are as good as fresh (derivation-scoped) variables. Syntactically, this fact is represented by the renaming lemma (Lemma 23). At the semantic level, this corresponds to creating a new Kripke model from a given one such that the semantics remains nearly identical.

Definition 15 *Given a Kripke model* $\mathcal{K} = (\mathcal{W}, \leq, \Vdash, \mathcal{D}, V)$*, a constant* $c \in \mathtt{nat}$*, and a value* $d \in \mathcal{D}$*, we define a new Kripke model* $\mathcal{K}_{c,d} := (\mathcal{W}, \leq, \Vdash, \mathcal{D}, V_{c,d})$*, where*

$$V_{c,d}(c') := \begin{cases} d & \text{if } c = c', \\ V(c') & \text{otherwise.} \end{cases}$$

That is, $\mathcal{K}$ and $\mathcal{K}_{c,d}$ differ only in the evaluation of the constant c. Consequently, we can present the following lemma:

Lemma 16 (Forcing with fresh constants) *Given a pseudo-formula* A *and a constant* c*, if* c *does not occur in* A*, then the following holds for all* $w \in \mathcal{W}$ *and* $d \in \mathcal{D}$*:*

$$w \Vdash_{\mathcal{K}} A[\eta] \iff w \Vdash_{\mathcal{K}_{c,d}} A[\eta]$$

under the condition that $\mathtt{BV}(A) \subseteq \mathit{dom}(\eta)$*. (Note that* $\mathtt{BV}(A) \subseteq \mathit{dom}(\eta)$ *trivially holds when* A *is a well-formed formula.)*

Now, $\omega \Vdash_{\mathcal{K}_{c,d}} \Gamma$ holds by Lemma 16. Consequently, by induction hypothesis, we also have $w \Vdash_{\mathcal{K}_{c,d}} (A\,[\,x \setminus c\,])[\eta]$; hence, (6) follows:

$$\begin{aligned} w \Vdash_{\mathcal{K}_{c,d}} (A\,[\,x \setminus c\,])[\eta] &\iff w \Vdash_{\mathcal{K}_{c,d}} A[(x, d) :: \eta] \\ &\iff w \Vdash_{\mathcal{K}} A[(x, d) :: \eta], \end{aligned} \tag{7}$$

where the equivalence in (7) is obviously true.

5.4 Universal Completeness

The universal Kripke model $\mathcal{U}$ consists of contexts as worlds, the sub-context relation $\subseteq$, and the provability for atomic sentences, and $\mathtt{term}$, the set of well-formed terms, as the the constant domain:

Definition 17 (Universal Kripke Model) $\mathcal{U} = (\mathtt{context}, \subseteq, \Vdash_{\mathcal{U}}, \mathtt{term}, V)$ *where*

$$V(c) = c, \qquad V(f)(t_1, t_2) = f\, t_1\, t_2\,.$$

Furthermore, $\Gamma \Vdash_{\mathcal{U}} P\, t$ *iff* $\Gamma \vdash P\, t$ *holds.*

Note that in the universal model $\mathcal{U}$, the interpretation of pseudo-terms is substitution; given a term $t \in \mathtt{pterm}\, m$ and an association $\eta = (x_1, u_1), ..., (x_n, u_n)$, where $u_i \in \mathtt{term}$, we have $t[\eta] = t\,[\,\eta \triangleright nil\,]$. Universal Completeness, as stated below, indicates that we have the similar correspondence between forcing and deduction:

Theorem 18 (Universal Completeness) *Let* $A \in \mathtt{pformula}\, m$, $\Gamma \in \mathtt{context}$, *and* η *be an association. Then,* $\Gamma \Vdash_{\mathcal{U}} A[\eta]$ *implies* $\Gamma \vdash A\,[\,\eta \triangleright nil\,]$.

Universal Completeness can be proved simultaneously with the following fact:

> If $\forall (C : \mathtt{formula})\, (\Gamma' : \mathtt{context})\, (\Gamma \subseteq \Gamma' \ \wedge\ \Gamma' \mid A\,[\,\eta \triangleright nil\,] \vdash C \ \Rightarrow\ \Gamma' \vdash C)$ holds, so does $\Gamma \Vdash_{\mathcal{U}} A[\eta]$.

Using this fact, one can easily show that $\Gamma \Vdash_{\mathcal{U}} \Gamma$ holds; consequently, we have the following:

Theorem 19 (Completeness) *Let* A *be a formula and* Γ *a context. If, in any Kripke model* $\mathcal{K}$, $w \Vdash A$ *follows from* $w \Vdash \Gamma$, *then* $\Gamma \vdash A$.

Remark 20 (With derivation-scoped variables) *Even with derivation-scoped (i.e., free) variables, the domain of the universal Kripke model remains* $\mathtt{term}$, *the set of well-formed terms with possible occurrences of derivation-scoped variables.*

Simultaneous substitution is of the form $A\,[\,\rho, \eta \triangleright \ell\,]$, *where* ρ *and* η *are responsible for derivation-scoped and formula-scoped variables, respectively. Correspondingly, the forcing relation has the form* $\omega \Vdash_{\mathcal{K}} A[\rho, \eta]$. *Finally, (Universal Completeness) changes slightly:*

> *Given* $A \in \mathtt{pformula}\, m$, $\Gamma \in \mathtt{context}$, *and* $\rho, \eta \in \mathtt{list}\,(\mathtt{nat} * \mathtt{term})$, *if* $\mathtt{FV}(A) \subseteq \mathsf{dom}(\rho)$ *and* $\Gamma \Vdash_{\mathcal{U}} A[\rho, \eta]$, *then* $\Gamma \vdash A\,[\,\rho, \eta \triangleright nil\,]$.

Here, $\mathtt{FV}(A)$ *is the set of derivation-scoped variables occurring in* A. *Note that Theorem 18 becomes a special case, where* $\mathtt{FV}(A) = \varnothing$.

5.5 Normalization by Evaluation

A combination of completeness and soundness leads to cut-admissibility. Let us assume that both $\Gamma \mid A \vdash B$ and $\Gamma \vdash A$ hold. Then, by (Soundness) $\Gamma \Vdash_{\mathcal{U}} A$ and $\Gamma \Vdash_{\mathcal{U}} B$ hold. Consequently, (Universal Completeness) implies that $\Gamma \vdash B$ holds in LJT.

Theorem 21 (Cut-admissibility) *Let A, B be formulae and Γ a context. Then, (Cut) is admissible in* LJT*:*

$$\frac{\Gamma \mid A \vdash B \quad \Gamma \vdash A}{\Gamma \vdash B}\ (Cut)$$

Because (Cut) is a semantically sound rule, a composition of (Soundness) and (Universal Completeness) normalize any proof with (Cut) to a cut-free proof. Moreover, a program extraction (which is available in Coq) from the composition would provide a function program that produces a cut-free proof from a deduction with (Cut).

6 Exists-Fresh quantification, a variable binding

One of primary issues addressed in our work is the formal handling of $\forall$-quanticiation. This includes the following:

- Formalization of a proof system with adequate treatment of the freshness condition in the $\forall$ right introduction rule (see Figure 4);
- Statement and proof of a weakening lemma for this proof system, which preserves the freshness condition of derivations (see Lemma 22 below);
- Ensuring well-formed expressions (see Lemma 3);
- Definition of a notion of association of the variables occurring below a binder (see Remark 13);
- Characterization of a set of terms that will serve as standard model for the completeness proof (we have to ensure that any variable used in a binder avoids the variables in terms) (see Remark 20).

The second point depends heavily on the first point, i.e., the representation style of $\forall$ right quantification, while the others are not related. In this section, we discuss at some length the issue of adequate formal representations of quantification rules.

For the formal representation of $\forall$ right quantification, we use the so-called (Exists-Fresh) style because we believe it is the closest approach to the pen-and-paper representation. Indeed, the rule ($\mathit{Exists\text{-}Fresh}$-$\forall_R$) reflects the intuition that, if the premise holds for *some* c that does not occur free in Γ nor in $\forall x\, A$, c should not be affected by any operation during the deduction, and therefore, it should be possible for an arbitrary term t that $\Gamma \vdash A(t)$ holds.

Weakening and renaming There is a well-known issue about the (Exists-Fresh) style. It provides *too weak* an induction principle. For example, let us try to prove the weakening lemma below in an intuitive ways.

Lemma 22 (Weakening) *Let A, C be formulae and Γ, Γ' contexts such that $\Gamma \subseteq \Gamma'$.*

1. *$\Gamma \vdash A$ implies $\Gamma' \vdash A$.*
2. *$\Gamma\,; A \vdash C$ implies $\Gamma'\,; A \vdash C$.*

When proving this lemma by induction on the given deduction rules, in the case for ($\mathit{Exists\text{-}Fresh}$-$\forall_R$), we are given a derivation ending with

$$\frac{\Gamma \vdash A\,[\,x \setminus c\,] \quad \text{for some } c \notin \mathtt{BV}(A, \Gamma)}{\Gamma \vdash \forall x\, A}$$

and an induction hypothesis

$$\Gamma' \vdash A\,[\,x \setminus c\,]$$

for an arbitrary Γ' such that $\Gamma \subseteq \Gamma'$. Now, we must conclude that $\Gamma' \vdash \forall x\, A$. However, it is noteworthy that we cannot apply the (*Exists-Fresh*-$\forall_R$) directly because we do not know whether $c \notin \mathtt{BV}(\Gamma')$. To ensure the freshness of the instance c, we can choose another fresh constant d such that $d \notin \mathtt{BV}(\Gamma')$, expecting that the following holds:

$$\Gamma' \vdash A\,[\,x \setminus d\,]\,.$$

which follows from the renaming lemma.

Lemma 23 (Renaming) *Let A, C be formulae and Γ a context. Assume c is a constant such that $c \notin \mathtt{BV}(C, \Gamma)$.*

1. *$\Gamma \vdash A\,[\,x \setminus c\,]$ implies $\Gamma \vdash A\,[\,x \setminus d\,]$.*
2. *$\Gamma\,; A\,[\,x \setminus c\,] \vdash C$ implies $\Gamma\,; A\,[\,x \setminus d\,] \vdash C$.*

When proving (Renaming) by induction on the deduction, the following case is critical. Let us assume that the deduction ends with an application of (*Exists-Fresh*-$\forall_R$) as follows:

$$\frac{\Gamma \vdash (A\,[\,x \setminus c\,])\,[\,y \setminus c'\,] \quad \text{for some } c' \notin \mathtt{BV}(A\,[\,x \setminus c\,], \Gamma)}{\Gamma \vdash \forall y\,(A\,[\,x \setminus c\,])}$$

where $x \neq y$ and $c \neq c'$. Using induction hypothesis, we can show that

$$\Gamma \vdash (A\,[\,x \setminus d\,])\,[\,y \setminus c'\,]\,. \tag{8}$$

However, we cannot apply (*Exists-Fresh*-$\forall_R$) because we do not know whether $c' \notin \mathtt{BV}(A\,[\,x \setminus d\,], \Gamma)$. We are trapped in a recursive problem: We need (Renaming) in order to prove (Renaming).

It is very common in pen-and-paper work of proof theory to use proof-length induction to solve this problem. The proof-length of a derivation is the length of the longest path of its derivation tree. In this case, we can prove that $\Gamma \vdash B\,[\,x \setminus d\,]$ can be proved with the same proof-length as that of $\Gamma \vdash B\,[\,x \setminus c\,]$ for an arbitrary formula B. Therefore, we can replace c' in (8) with a totally fresh c'' such that the induction hypothesis can be applied. However, this solution requires relatively heavy infrastructure about proof-length.

Equivalence of three well-known quantification styles Another standard way to solve the aforementioned recursive problem is to strengthen the induction principle for deduction by changing (*Exists-Fresh*-$\forall_R$) to one of the following styles:

$$\frac{\Gamma \vdash A[x\backslash c] \quad \text{for all } c \notin \mathtt{OC}(\forall x\, A :: \Gamma)}{\Gamma \vdash \forall x\, A} \;\; (\textit{All-Fresh}\text{-}\forall_R)$$

or

$$\frac{\Gamma \vdash A[x\backslash c] \quad \text{for all } c \notin L}{\Gamma \vdash \forall x\, A} \;\; (\textit{Cofinite}\text{-}\forall_R)$$

where L denotes a finite set of constants.

The (All-Fresh) style is used in McKinna and Pollack [20, 21] and in Leroy's solution [19] to the POPLmark Challenge, while in Aydemir et al. [2], the efficiency of the cofinite

quantification style is clearly explained. Furthermore, it is proved in each paper that the typability of terms (provability of formulas in our work) remains the same.

Let LJT_a and LJT_c be variants of LJT where (*Exists-Fresh*-$\forall_R$) is replaced with (*All-Fresh*-$\forall_R$) and (*Cofinite*-$\forall_R$), respectively. In both LJT_a and LJT_c, (Weakening) can be proved easily by a simple induction on the deduction and (Renaming) is not necessary anymore. Let $\vdash_{\overline{a}}$ denote the derivation in LJT_a, and $\vdash_{\overline{c}}$, in LJT_c.

The equivalence of the three styles can be proved in a straightfoward manner by structural induction on the deduction, except for the following case:

$$\Gamma \vdash A \;\Rightarrow\; \Gamma \vdash_{\overline{a}} A \tag{9}$$

A naive approach would encounter the same problem as before, i.e., the induction hypothesis in the (*Exists-Fresh*-$\forall_R$) is too weak. McKinna and Pollak show (9) using the fact that *bijective renaming* respects $\vdash_{\overline{a}}$, which in our case, would appear as

$$\Gamma \vdash_{\overline{a}} A \;\Rightarrow\; \rho\,\Gamma \vdash_{\overline{a}} \rho\,A. \tag{10}$$

where $\rho\,(\cdot)$ stands for a bijective renaming of constants (cf. Section 5.2.1 in [21]). It is to be noted that renaming is a special form of substitution where constants are substituted by constants. Leroy [19] follows a similar approach using *swap* functions, which are special forms of bijective renamings.

Using simultaneous renaming Although the equivalence proof is relatively straightforward, we have to check whether we really need to strengthen the induction principle in order to prove that (Weakening) and (Renaming) hold in LJT. This is also related to the question whether the excursion to LJT_a or LJT_c is really necessary.

If we revisit the points where the intuitive proofs of (Weakening) and (Renaming) could not proceed further, we notice that (Weakening) needs (Renaming) and that (Renaming) in turn needs to be proved by using simultaneous renaming, as in (10). In other words, (Weakening) and (Renaming) could be proved together based on *simultaneous renaming.*

Theorem 24 (Generalized Weakening) *Let A, C be formulae, Γ, Γ' contexts such that $\Gamma \subseteq \Gamma'$, and ρ an arbitrary renaming.*

1. *$\Gamma \vdash A$ implies $\rho\,\Gamma' \vdash \rho\,A$.*
2. *$\Gamma\,; A \vdash C$ implies $\rho\,\Gamma'\,; \rho\,A \vdash \rho\,C$.*

Note that there are no side conditions on renaming ρ. Both claims can be proved by a simple simultaneous structural induction. Finally, (Weakening) and (Renaming) are special forms of (Generalised Weakening). Further, we remark that (9) can be proved in a similar way:

$$\Gamma \vdash A \;\Rightarrow\; \rho\,\Gamma \vdash_{\overline{a}} \rho\,A$$

where ρ denotes an arbitrary renaming.

7 Conclusion

We proposed a new first-order representation, called representation with locally traced names, of logical formal systems with variable binding. The main feature is that an extra syntax for well-formed terms and formulae are not necessary even if two sorts of variable names are used. In order to demonstrate the adequacy of our representation style, we formalized in Coq the soundness and completeness of intuitionistic first-order predicate logic with respect to a Kripke semantics. As a result, the cut-elimination follows based on normalization by evaluation (NBE), i.e., by the composition of completeness and soundness. We remark that the mechanized proofs are nearly identical to the informal proofs given by Herbelin and Lee [15].

In addition to the new representation style, we incorporated two more suggestions that helped reduce the basic infrastructure with respect to variable binding and substitution. First, merging derivation-scoped variables with constants enabled us to avoid several substitution issues. Second, using simultaneous subsitution and renaming, it was possible to avoid any type of length induction or excursion to other equivalent deduction systems. In particular, this point forced us to reinvestigate the role of the (Exists-Fresh) style of quantification, and from our experience, we conclude that the (Exists-Fresh) style is probably the best solution for the issue of quantification style.

In the future, we plan to focus on our approach when derivability is a part of the domain of the discourse. This will be the case if one wants to prove that the cut-free derivation obtained by semantic cut-elimination actually implements β-normalization and η-expansion over derivations, as shown by Coquand [7,8] in her semantic normalization proof for implication logic.

References

1. Ambler, S.J., Crole, R.L., Momigliano, A.: A definitional approach to primitivexs recursion over higher order abstract syntax. In: MERLIN. ACM (2003)
2. Aydemir, B.E., Charguéraud, A., Pierce, B.C., Pollack, R., Weirich, S.: Engineering formal metatheory. In: POPL, pp. 3–15. ACM Press (2008)
3. Berger, U., Eberl, M., Schwichtenberg, H.: Normalisation by Evaluation. In: Prospects for Hardware Foundations, *Lecture Notes in Computer Science*, vol. 1546, pp. 117–137. Springer (1998)
4. Berger, U., Schwichtenberg, H.: An inverse of the evaluation functional for typed lambda-calculus. In: LICS, pp. 203–211. IEEE Computer Society (1991)
5. de Bruijn, N.G.: Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. Indagationes Mathematicae **34**(5), 381–392 (1972)
6. Church, A.: Introduction to mathematical logic. Princeton University Press (1996). (First published in 1944 in the *Annals of Mathematics Studies*)
7. Coquand, C.: From Semantics to Rules: A Machine Assisted Analysis. In: CSL '93, *Lecture Notes in Computer Science*, vol. 832, pp. 91–105. Springer (1993)
8. Coquand, C.: A formalised proof of the soundness and completeness of a simply typed lambda-calculus with explicit substitutions. Higher-Order and Symbolic Computation **15**(1), 57–90 (2002)
9. Coquand, T.: An algorithm for testing conversion in type theory. In: Huet, G., Plotkin, G. (eds.) Logical Frameworks, pp. 255–279. Cambridge University Press (1991)
10. Geuvers, H.: Logics and Type Systems. Ph.D. thesis, University of Nijmegen (1993)
11. Gordon, A.D., Melham, T.F.: Five axioms of alpha-conversion. In: TPHOLs, *Lecture Notes in Computer Science*, vol. 1125, pp. 173–190. Springer (1996)
12. Harper, R., Licata, D.R.: Mechanizing metatheory in a logical framework. J. Funct. Program. **17**(4-5), 613–673 (2007)

13. Herbelin, H.: A Lambda-Calculus Structure Isomorphic to Gentzen-Style Sequent Calculus Structure. In: CSL '94, *Lecture Notes in Computer Science*, vol. 933, pp. 61–75. Springer (1994)
14. Herbelin, H.: Séquents qu'on calcule: de l'interprétation du calcul des séquents comme calcul de λ-termes et comme calcul de stratégies gagnantes. Ph.D. thesis, Université Paris 7 (1995)
15. Herbelin, H., Lee, G.: Forcing-Based Cut-Elimination for Gentzen-Style Intuitionistic Sequent Calculus. In: Ono, H., Kanazawa, M., de Queiroz, R.J.G.B. (eds.) WoLLIC, *Lecture Notes in Computer Science*, vol. 5514, pp. 209–217. Springer (2009)
16. Ilik, D., Lee, G., Herbelin, H.: Kripke models for classical logic. Ann. Pure Appl. Logic **161**(11), 1367–1378 (2010)
17. Kripke, S.: A Completeness Theorem in Modal Logic. J. Symb. Log. **24**(1), 1–14 (1959)
18. Kripke, S.: Semantical considerations on modal and intuitionistic logic. Acta Philos. Fennica **16**, 83–94 (1963)
19. Leroy, X.: A locally nameless solution to the poplmark challenge. Tech. Rep. 6098, INRIA (2007)
20. McKinna, J., Pollack, R.: Pure type systems formalized. In: TLCA, pp. 289–305 (1993)
21. McKinna, J., Pollack, R.: Some lambda calculus and type theory formalized. Journal of Automated Reasoning **23**(3-4), 373–409 (1999)
22. Sato, M., Pollack, R.: External and internal syntax of the lambda-calculus. J. Symb. Comput. **45**(5), 598–616 (2010)
23. Stoughton, A.: Substitution revisited. Theor. Comput. Sci. **59**, 317–325 (1988)
24. Troelstra, A.S., van Dalen, D.: Constructivism in Mathematics: An Introduction I and II, *Studies in Logic and the Foundations of Mathematics*, vol. 121, 123. North-Holland (1988)
25. Troelstra, A.S., van Dalen, D.: Constructivism in Mathematics Volume I, *Studies in Logic and the Foundations of Mathematics*, vol. 121. North-Holland (1988)
26. Urban, C.: Nominal techniques in isabelle/hol. J. Autom. Reasoning **40**(4), 327–356 (2008)