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

Formalizing Logical Metatheory

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 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 *globally*

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

bound variables (that are themselves bound in the right introduction rule of the universal quantification) are used for representing parametric derivations.[1] In this paper, *variables* stand for the (locally bound) variables such as x in $\forall x\, P(x)$, whereas *parameters* 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 variables and parameters. The main issue in this approach is the possible capture of a parameter by a variable during substitution of a term in an expression. A typical way of addressing this issue is to ensure that all the parameters are always distinct from the variables. The use of α-conversion makes this possible.

However, during the mechanical development of a formal metatheory, 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.

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 these studies here; we just refer the reader to some papers that include such discussion (cf. [2,3,7,12,13,23,27]).

Our approach is similar to McKinna-Pollack's locally-named representation [22, 23]: variables and parameters 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 main feature of our approach is that the sets of terms and formulae depend on a list of variables, called *a trace*, that might be used during the term or formula construction.[2] 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 with an empty trace.)

We remark that there are approaches with two sorts of variables that require no extra syntax for well-formedness. Sato and Pollack [25], 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. 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, one with parameters

[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, Coquand [8] 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

[2] It is not a new idea to use indexed types to carry around the possible freely occurring names, cf. [4,1,21]. However, our work is a specific and practical realization of this idea.

and the other without parameters. The difference between the two styles will be explained in Section 3. The structures of both mechanizations are nearly identical. This paper is based on the version without parameters. The Coq proof scripts are available online.[3]

Outline of the paper The main features of the paper are summarized in Section 2. In Section 3, we explain parameters and 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 parameters. We investigate the roles of constants and parameters, and we show that a fresh constant can play exactly the same role as a fresh parameter. In other words, there is no real difference between parameters and constants. The basic idea is as follows: Assume we have a Gentzen-style derivation for $\Gamma \vdash A$ in which parameters $a_1, ..., a_n$ could possibly occur. Metatheoretically, this means that $\forall a_1 \cdots a_n[\Gamma \vdash A]$ holds. That is, the derivation is valid for arbitrary, but *fixed* values $a_1, ..., a_n$.

The consequences are manifold: There is no concern regarding variable capture during substitution, substitution is not required for parameters, and a more compact formalization (simpler definitions and shorter proof terms) can be achieved.

Second, even if two sorts of variable names are used, no extra syntax for well-formed terms and formulae is necessary. This significantly reduces the syntax part during formalization.

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

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

Here, $[\,c\,/\,x\,]\,A$ denotes the well-formed formula resulting from A by substituting a constant c for x, and $OC(\Delta)$ denotes the set of constants occurring in the context Δ. (Note that we use fresh constants as fresh parameters.)

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.

Fourth, 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).

Further, they play an essential role in showing that the Exists-Fresh style is adequate for dealing with quantification.[4] And the equivalence of three well-known quantification styles (Exists-Fresh, Cofinite, and All-Fresh styles) can be easily proved. Although the equivalence is already known, our proof reveals certain new aspects of

[3] Please visit `http://formal.hknu.ac.kr/Kripke/`.

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

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.

Fifth, the cut-elimination is based on normalization by evaluation for intuitionistic first-order predicate logic: A composition of completeness and soundness leads to cut-elimination. Because all the proofs are constructive, our work can be seen as an extension of Coquand's work [8,9], where first-order propositional logic is the main object. Refer to Berger et al. [5] and Berger and Schwichtenberg [6] for more details regarding normalization by evaluation.

3 Variables, constants, and α-conversion

Parameters and 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 irrelevant.

Second, the $\forall$-quantifiers are instantiated by terms that do not contain 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 parameters 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 parameters.

However, the need for α-conversion over parameters and instantiation of parameters will arise if derivability is lifted as the object of the domain of discourse as in Coquand [8,9]. She showed that the cut-free derivation obtained by semantic cut-elimination actually implements β-normalization and η-expansion over derivations.

Constants as parameters The intended meaning of a derivation in which parameters occur as in $A(x) \vdash B(x)$ is the collection of all derivations obtained by instantiating the variables by arbitrary, but fixed terms. However, parameters are not instantiated by any rule. Further, as we will see in Lemma 16 and Lemma 24, we can replace a fresh constant with arbitrary terms when it does not occur in any assumptions.

All these facts suggest that parameters 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 parameters as constants at the syntactic level is that substitution is needed only for variables.

To do this, 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 also noteworthy that even if we formally introduce parameters, the substitution for parameters do not play any role (cf. the Coq formalization with parameters).

On the purpose of α-conversion We remark that terms that differ only by the names of variables are not considered to be equivalent. This may be surprising as it departs from the common usage of reasoning modulo α-conversion. For example, in McKinna and Pollack [22,23], where the metatheory of Pure Type systems is formalized, α-conversion is necessary in showing the Church-Rosser property.

In our work, where derivability is a predicate and not a part of the domain of the discourse, it is harmless because for any derivation that would consider some formulae modulo α-conversion, there is another one that differs from the original only by the names of 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 exist $\Gamma' \overset{\alpha}{\equiv} \Gamma$ and $A' \overset{\alpha}{\equiv} A$ such that $\Gamma' \vdash A'$*

An informal[5] proof is by induction. Most of all, 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 $\forall y\, A'(y) \overset{\alpha}{\equiv} \forall x\, A(x)$. $\square$

4 Representation with locally traced names

The usage of two sorts of variable names was first implemented by McKinna and Pollack [22,23] to formalize the Pure Type System (PTS) metatheory, following the suggestion by Coquand [10]. However, there are certain limitations. For example, not all syntactic expressions are meaningful because variables could occur unbound. This is also the case, even though parameters are replaced by constants. Therefore, it is usually required to provide an extra syntax for the definition of well-formed expressions.[6]

Here, we introduce a new first-order approach, where no extra syntax is necessary for well-formed expressions, even though variables and parameters are syntactically distinguished. The explanation will be provided by demonstrating how to formalize the relationship between intuitionistic first-order predicate logic and 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 [8] and we found it interesting to try an alternative approach.

[5] We do not provide any formal work regarding α-conversion.

[6] In McKinna and Pollack [22,23], well-formed expressions are called *variable-closed* while they are called *locally closed* in Aydemir et al. [3].

Let $\texttt{name} := \texttt{nat}$ and $m \in \texttt{list name}$.

Pseudo-terms:

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

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

Pseudo-formulae:

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

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

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

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

Occurrence of constants:

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

Occurrence of variables:

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

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

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 variables and constants.[7] `Bvar` stands for the denotation of 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 variables. For simplicity, we assume two denumerable and decidable sets: `predicate` of unary predicates and `function` of binary functions.[8] Note that a set X is *decidable* if, constructively, $\forall u, v \in X\,(u = v \vee u \neq v)$, otherwise said, if there exists a decision function f from $X \times X$ such that $u = v \leftrightarrow f(u, v) = 0$. 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 set-theoretic manner: A list ℓ is a sublist of another list k if ℓ is a subset of k when regarding them 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. The notation $\in$ and $\notin$ are used with respect to the being-in-a-list *relation. nil stands for the empty list, and $x_1, ..., x_n$ stands for the list $x_1 :: \cdots :: x_n :: nil$.*

[7] In fact, any decidable, denumerably infinite set can be used instead of natural numbers. We use natural numbers for simplicity.

[8] We remark that our assumption causes any loss of generality because functions or predicates of other arities can be represented by using binary function symbols.

Terms and formulae as inductive families Given a list of names $m \in \texttt{list name}$, $\texttt{pterm}\, m$ and $\texttt{pformula}\, m$ denote the set of pseudo-terms and pseudo-formulae, respectively (see Figure 1). Intuitively, the list m, which we call *trace*, collects the variables that possibly occur unbound. We use the notation "*pseudo-*" because some variables can occur unbound while they are supposed to be bound by a ∀-quantifier. (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 the construction of pseudo-terms or -formulae:

Lemma 3 *Let $e \in \texttt{term}\, m$ or $e \in \texttt{pformula}\, m$. Then, $\texttt{OV}(e) \subseteq m$. In particular, $\texttt{OV}(e)$ 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. $\texttt{pformula}\, nil$):

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

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 pen-and-paper style. 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 necessary to check whether a pseudo-formula occur in a context, which is a list of (well-formed) 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 the internal representation of Coq terms 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 variables, we superimpose de Bruijn indices and names so that α-equivalence (implicit in Theorem 5 below) is easily checked.

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[9] to the standard one from the Coq library for lists: Given a decidable set X and $m \in \texttt{list}\, X$,

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

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

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

Let m and ℓ be traces.

1. A partial function $\texttt{treloc}^{m,\ell} : \texttt{pterm}\,m \to \texttt{pterm}\,\ell$, defined only for terms t such that $\texttt{OV}(t) \subseteq \ell$, is recursively defined: ($\texttt{treloc}$ denotes $\texttt{treloc}^{m,\ell}$, for simplicity.)

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

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

2. A partial function $\texttt{freloc}^{m,\ell} : \texttt{pformula}\,m \to \texttt{pformula}\,\ell$, defined only for formulae A such that $\texttt{OV}(A) \subseteq \ell$, is recursively defined: ($\texttt{freloc}$ denotes $\texttt{freloc}^{m,\ell}$, for simplicity.)

$$\begin{aligned} \texttt{freloc}(P\,t) &= P\,(\texttt{treloc}(t)) \\ \texttt{freloc}(A \to B) &= \texttt{freloc}(A) \to \texttt{freloc}(B) \\ \texttt{freloc}(\forall x\,C) &= \forall x\,(\texttt{freloc}(C)) \end{aligned}$$

   where $\texttt{freloc}(C) = \texttt{freloc}^{x::m,x::\ell}(C)$ depends on the fact that $\texttt{OV}(C) \subseteq x :: \ell$ which trivially follows from $\texttt{OV}(\forall x\,C) \subseteq \ell$.

Fig. 2 Trace relocation

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

$$\forall (p, q : x \in m)\,[p = q]\,.$$

Now, the syntactic decidability follows easily.

Theorem 5 (Syntactic decidability) *Let m be a trace.*

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

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

For any pseudo-term $t \in \texttt{pterm}\,m$, the set $k = \texttt{OV}(t) \subseteq m$ is the canonical and smallest trace such that $t \in \texttt{pterm}\,k$. We use this property to relocate the type of t using a *homomorphic* function from $\texttt{pterm}\,m$ to $\texttt{pterm}\,\ell$ when $\texttt{OV}(t) \subseteq \ell$. The relocation 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 11.) We emphasize that trace relocation is necessary to deal with substitution (cf. Figure 3).

The relocation function presented in Figure 2 is based on the fact that the being-in-a-list part is inessential so long as the trace contains all the variables needed for the construction of an expression e, i.e., $\texttt{OV}(e)$. Note also that $\texttt{treloc}^{m,\ell}(t)$ 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. This indicates that repeated application of relocation is equivalent to a single application and that the syntactic equality between two expressions is preserved during trace relocation.

In the following lemmata, we omit necessary variable conditions for a simple presentation.

Let m, ℓ be traces and $\eta = (x_1, u_1), ..., (x_n, u_n)$ be an association, where $u_i \in \texttt{term}$. Further, $t \in \texttt{pterm}\, m$ and $A \in \texttt{pformula}\, m$.

1. $[\,\ell \Uparrow \eta\,]\, t \in \texttt{pterm}\, \ell$ is recursively defined by:

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

$$[\,\ell \Uparrow \eta\,]\, (\texttt{Cst}\, c) = \texttt{Cst}\, c$$

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

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

2. $[\,\ell \Uparrow \eta\,]\, A \in \texttt{pformula}\, \ell$ is recursively defined by:

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

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

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

Fig. 3 Simultaneous substitution with destination

Lemma 6 (Idempotence) *Assume* $e \in \texttt{pterm}\, m$ *or* $e \in \texttt{pformula}\, m$. *Then,*

$$\texttt{reloc}^{\ell,k}(\texttt{reloc}^{m,\ell}(e)) = \texttt{reloc}^{m,k}(e)$$

where `reloc` *denotes* `treloc` *or* `freloc` *depending on* e.

The following lemma says that relocation functions preserve equality.

Lemma 7 (Equality preservation) *Assume* $e, e' \in \texttt{pterm}\, m$ (*or* $e, e' \in \texttt{pformula}\, m$). *Then,* $\texttt{reloc}^{m,\ell}(e) = \texttt{reloc}^{m,\ell}(e')$ *as soon as* $e = e'$. *Here,* `reloc` *denotes* `treloc` *or* `freloc` *depending on* e, e'.

The overhead for dealing with relocation is very small. In addition to the two definitions for `treloc` and `freloc`, we needed only 14 lemmata, which are all proved in several lines.

Substitution with destination The definition of a substitution requires special consideration for 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 find that simultaneous substitution provides a convenient way in establishing the relationship, see the Soundness (Theorem 14) and the Universal Completeness (Theorem 18). In Section 6, we will also see that *simultaneous renaming* of constants plays an important role.

The simultaneous substitution of finitely many terms for variables in a pseudo-term or a -formula is defined by a structural recursion as in Figure 3. Only well-formed terms are substituted because substituting pseudo-terms could cause capture of variables.

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

Let e be a pseudo-term or a -formula, ℓ a trace, $\eta = (x_1, u_1), ..., (x_n, u_n)$ an association, where $u_i \in \texttt{term}$. Then $[\,\ell \Uparrow \eta\,]\,e$ denotes the simultaneous substitution of u_i for x_i, $1 \leq i \leq n$, in e. The single substitution $[\,\ell \Uparrow u\,/\,x\,]\,e := [\,\ell \Uparrow (x, u)\,]\,e$ is then a special case. Furthermore, we write $[\,u\,/\,x\,]\,e$ when $\ell = \mathit{nil}$ for better readability.

The type of the resulting expression depends on ℓ, not on the type of e. Intuitively, ℓ is the list of variables for which no substitution is allowed. The role of ℓ is well explained by the the abstraction case (3) in Figure 3, where ℓ is extended by x in order to forbid any substitution for x. Note that all insignificant variables are simply ignored by assigning them $\texttt{Cst}\,0$. They are variables occurring neither in ℓ nor in $x_1, ..., x_n$, cf. (2).

In particular, we have $[\,\mathit{nil} \Uparrow \eta\,]\,t \in \texttt{term}$ and $[\,\mathit{nil} \Uparrow \eta\,]\,A \in \texttt{formula}$ independent of the type of t or A. Consequently, we can use more intuitive definitions and proofs. For example, the left introduction rule of universal quantification in Figure 4 can be formalized in the following form

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

without referring to the well-formedness of $[\,t\,/\,x\,]\,A$. Another case is the Universal Completeness (Theorem 18), in which simultaneous substitution with destination is effectively used for a natural correspondence between semantics and syntax. A detailed explanation is given in Remark 19.

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

Lemma 8 (Substitution Lemma) *Given two traces m and ℓ, let $e \in \texttt{pterm}\,m$ or $e \in \texttt{pformula}\,m$, $u \in \texttt{term}$, and η an association. Then we have*

$$[\,\ell \Uparrow u\,/\,y\,]\,([\,y :: \ell \Uparrow \eta\,]\,e) = [\,\ell \Uparrow (y, u) :: \eta\,]\,e\,.$$

Lemma 9 (Relocation and substitution) *Given three traces k, m, and ℓ, let $e \in \texttt{pterm}\,m$ or $e \in \texttt{pformula}\,m$ and $\texttt{OV}(e) \subseteq k$. Then we have*

$$[\,\ell \Uparrow \eta\,]\,(\texttt{reloc}^{m,k}(e)) = [\,\ell \Uparrow \eta\,]\,e\,,$$

where `reloc` *is* `treloc` *or* `freloc` *depending on e. That is, relocation has no impact on substitution.*

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

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

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

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

Fig. 4 Cut-free LJT

than the so-called natural deduction. However, natural deduction has already been used in Coquand [8] 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 the vertical bar "|" in the antecedent is called *stoup*, and it contains the principal formula (Hauptformel) of the corresponding rule.

Herbelin [14,15] and Mints [24] showed that there is 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 [26, Ch. 1]), this amounts to giving names to formulae (see Geuvers [11] for a discussion).

For our work, however, Curry-Howard-de Bruijn correspondence itself is not on the program. Once again. this is because derivability is a predicate and not a part of the domain of the discourse. Furthermore, contexts can be regarded as finite sets although we used lists of formulae to represent them.

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 `formula` = `pformula` *nil*. Note that typing

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

and $\Vdash$ is a relation between $\mathcal{W}$, `predicate`, and $\mathcal{D}$ such that

$$\text{if } (w \leq w' \text{ and } w \Vdash P\,d) \text{ holds, then } w' \Vdash P\,d\,.$$

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

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

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

Here $\eta(x) = d$ if (x, d) is the first occurrence 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

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 [22,23], 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 $Ok(\Delta)$ denotes that all formulae in the context Δ 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 lemmata about well-formed formulae and contexts (cf. McKinna and Pollack [22,23] and Aydemir et al. [3]).

5.2 Kripke semantics

Kripke semantics was created in the late 1950s and early 1960s by Saul Kripke [18,19]. 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 [26] and Ilik et al. [17]). Here, we use the conventional Kripke model adopted by Troelstra and van Dalen [26].

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 relation between worlds, predicates, and domain elements (cf. Figure 5).

Note that the interpretation of pseudo-terms is made total by ignoring insignificant variables. Furthermore, the being-in-a-list part of a term is simply thrown away. Consequently, the trace relocation has no impact on the Kripke semantics.

Let a Kripke model $\mathcal{K} = (\mathcal{W}, \leq, \Vdash, \mathcal{D}, V)$ and an association $\eta \in \texttt{list}\,(\texttt{name} * \mathcal{D})$ be given.

Theorem 11 (Relocation-irrelevance) *Let m be a trace, $t \in \texttt{pterm}\,m$, and $A \in \texttt{pformula}\,m$.*

1. $t[\eta] = \texttt{treloc}(t)[\eta]$.
2. $w \Vdash A[\eta]$ *if and only if* $w \Vdash \texttt{freloc}(A)[\eta]$.

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

Lemma 12 (Monotonicity) *Let m be a trace. Given a pseudo-formula $A \in \texttt{pformula}\,m$, 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:*

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

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 [16]).

5.3 Soundness

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

Theorem 14 (Soundness) *Let $A :: C :: \Gamma$ be a context.*

1. *Suppose $\Gamma \vdash C$ holds. Then for any Kripke model $\mathcal{K} = (\mathcal{W}, \leq, \Vdash_{\mathcal{K}}, \mathcal{D}, V)$ and any $w \in \mathcal{W}$, if $w \Vdash_{\mathcal{K}} \Gamma$ holds, so does $w \Vdash_{\mathcal{K}} C[nil]$.*
2. *Suppose $\Gamma \mid A \vdash C$ holds. Then for any Kripke model $\mathcal{K} = (\mathcal{W}, \leq, \Vdash_{\mathcal{K}}, \mathcal{D}, V)$ and any $w \in \mathcal{W}$, if $w \Vdash_{\mathcal{K}} \Gamma$ and $w \Vdash_{\mathcal{K}} A[nil]$ hold, so does $w \Vdash_{\mathcal{K}} C[nil]$.*

If we had included parameters, the soundness proof would be a simple mutual induction on the derivation, cf. Herbelin and Lee [16]. However, because we use constants instead, the (*Exists-Fresh*-$\forall_R$) rule requires more attention.

Suppose that $\Gamma \vdash \forall x\, A$ follows from $\Gamma \vdash [\,c\,/\,x\,]\,A$ for a constant $c \notin \texttt{OC}(A, \Gamma)$ that $w \Vdash \Gamma$ holds. Then, given an arbitrary $d \in \mathcal{D}$, we have to show that

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

holds. At this point, the premise of (*Exists-Fresh*-$\forall_R$) seems to provide too weak an induction hypothesis. That is, 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 parameters. Syntactically, this fact is represented by the renaming lemma (Lemma 24). 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, 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*

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

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 any Kripke model $\mathcal{K} = (\mathcal{W}, \leq, \Vdash, \mathcal{D}, V)$, any $w \in \mathcal{W}$, and any $d \in \mathcal{D}$:*

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

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

Now, we use Lemma 16 to show that $\omega \Vdash_{\mathcal{K}_{c,d}} \Gamma$. Consequently, by induction hypothesis, we also have $w \Vdash_{\mathcal{K}_{c,d}} ([\,c\,/\,x\,]\,A)[nil]$. Finally, we can prove (6):

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

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 formulae, and `term` as the constant domain:

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

$$V_{\mathcal{U}}(c) = c \quad \textit{and} \quad V_{\mathcal{U}}(f)(t_1, t_2) = f\, t_1\, t_2\,.$$

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 corresponds to 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 $t[\eta] = [\,nil \Uparrow \eta\,]\,t$. The Universal Completeness, as stated below, indicates that we have a 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 $\Gamma \vdash [\,nil \Uparrow \eta\,]\,A$.*

Remark 19 *We emphasize that simultaneous substitution enables us to have this natural correspondence between syntax and semantics. Note also that A is an arbitrary pseudo-formula, i.e., no well-formedness is assumed. This implies that the theorem can be proved by a simple structural induction on A. This is a point where simultaneous substitution with destination plays an important role.*

The Universal Completeness can be proved by a mutual induction with the following fact:

> If $\forall (C : \texttt{formula})\, (\Gamma' : \texttt{context})\, (\Gamma \subseteq \Gamma' \ \wedge\ \Gamma' \mid [\,nil \Uparrow \eta\,]\, A \vdash C \ \Rightarrow\ \Gamma' \vdash C)$ 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 20 (Completeness) *Let A be a formula and Γ a context. If, for any Kripke model $\mathcal{K} = (\mathcal{W}, \leq, \Vdash, \mathcal{D}, V)$ and any $w \in \mathcal{W}$, $w \Vdash A$ follows from $w \Vdash \Gamma$, then we have $\Gamma \vdash A$.*

Remark 21 (With parameters) *Even if we had considered parameters, the domain of the universal Kripke model remains* `term`*, the set of well-formed terms with possible occurrences of parameters.*

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

> *Given $A \in \texttt{pformula}\, m$, $\Gamma \in \texttt{context}$, and two associations ρ, η, if $\texttt{OP}(A) \subseteq \mathsf{dom}(\rho)$ and $\Gamma \Vdash_{\mathcal{U}} A[\rho, \eta]$, then $\Gamma \vdash [\,nil \Uparrow \rho, \eta\,]\, A$.*

Here, $\texttt{OP}(A)$ is the set of parameters occurring in A. Note that Theorem 18 becomes a special case, where $\texttt{OP}(A) = \varnothing$.

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 $\Gamma \Vdash_{\mathcal{U}} A$, hence $\Gamma \Vdash_{\mathcal{U}} B$ holds by Soundness again. Consequently, $\Gamma \vdash B$ holds by Universal Completeness.

Theorem 22 (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) normalizes any proof with (Cut) to a cut-free proof. [10]

[10] A program extraction (which is available in Coq) from the composition would provide a functional program that produces a cut-free proof from a deduction with (Cut). We believe that the normalization follows the reduction semantic of LJT.

6 Exists-Fresh quantification, a variable binding

One of primary issues addressed in our work is the formal handling of $\forall$-quantification. 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 23 below);
- Ensuring well-formed expressions (see Lemma 3);
- 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 21).

The second point is closely related to the first point, i.e., to 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 (*Exists-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 way.

Lemma 23 (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 (*Exists-Fresh*-$\forall_R$), we are given a derivation ending with

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

and an induction hypothesis

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

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

$$\Gamma' \vdash [\,d\,/\,x\,]\,A\,. \tag{8}$$

This, however, would require renaming of constants.

(Simultaneous) Renaming is a kind of (simultaneous) substitution where constants are replaced with constants. In the following, $\rho = (c_1, d_1), ..., (c_n, d_n)$ stands for a simultaneous renaming. Given a pseudo-formula $A \in \mathtt{pformula}\, m$, $\rho\, A$ stands for a pseudo-formula in `pformula` m where each constant c_i occurring in A is simultaneously replaced with d_i. For a context Γ, $\rho\,\Gamma$ is canonically defined. Note that $[\,d\,/\,x\,]\,A \equiv \rho\,([\,c\,/\,x\,]\,A)$, where $\rho = (c, d)$.

Lemma 24 (Renaming) *Let A, C be formulae and Γ a context. Assume c is a constant such that $c \notin \mathtt{OC}(C, \Gamma)$. Given an arbitrary constant d, if $\rho = (c, d)$, then we have:*

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

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:

$$\frac{\Gamma \vdash [\,c'\,/\,y\,]\,B \quad \text{for some } c' \notin \mathtt{OC}(B, \Gamma)}{\Gamma \vdash \forall y\,B}$$

where $c \neq c'$. Using induction hypothesis, we can show that, for $\rho = (c, d)$,

$$\Gamma \vdash [\,c'\,/\,y\,]\,(\rho\,B)\,. \tag{9}$$

However, we cannot apply (*Exists-Fresh*-$\forall_R$) because we do not know whether $c' \notin \mathtt{OC}(\rho\,B, \Gamma)$. We would need an extra call to (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 $\Gamma \vdash \rho\,B$ can be proved with the same proof-length as that of $\Gamma \vdash B$ for an arbitrary formula B. Therefore, we can replace c' in (9) 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 problem is to strengthen the induction principle for deduction by changing (*Exists-Fresh*-$\forall_R$) to one of the following styles:

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

or

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

where L is a finite set of constants.

The (All-Fresh) style is used in McKinna and Pollack [22,23] and in Leroy's solution [20] to the POPLmark Challenge, while in Aydemir et al. [3], the efficiency of the cofinite quantification style is well explained. Furthermore, it is proved in each paper that the typability of terms (provability of formulae 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 $\vdash_a$ denote the derivation in LJT_a, and $\vdash_c$, in LJT_c.

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

$$\Gamma \vdash A \;\Rightarrow\; \Gamma \vdash_{\overline{a}} A \tag{10}$$

A naive approach would encounter the same problem as before, i.e., the induction hypothesis in the (*Exists-Fresh*-$\forall_R$) is too weak. Following McKinna and Pollak, we could show (10) using the fact that *bijective renaming* respects $\vdash_{\overline{a}}$:

$$\Gamma \vdash_{\overline{a}} A \;\Rightarrow\; \rho\,\Gamma \vdash_{\overline{a}} \rho\,A. \tag{11}$$

where $\rho\,(\cdot)$ stands for a *bijective* renaming of constants (cf. Section 5.2.1 in [23]). Leroy [20] follows a similar approach using *swap* functions, which are special forms of bijective renaming.

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 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 (11). This indicates that (Weakening) and (Renaming) could be proved together based on *simultaneous renaming.*

Theorem 25 (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 the renaming ρ, i.e., no bijectivity is required. Both claims can be proved by a simple mutual structural induction. Finally, (Weakening) and (Renaming) are special forms of (Generalized Weakening). Further, we remark that (10) 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 [16].

In addition to the new representation style, we incorporated two more suggestions that helped us reduce the basic infrastructure with respect to variable binding and substitution. First, merging parameters with constants enabled us to avoid several substitution issues. Second, using simultaneous substitution 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 first-order structure quantification. From our experience, we conclude that the (Exists-Fresh) style is probably the best solution for the issue of quantification style. Best in the sense that it is closest to the pen-and-paper style and that no functional types are involved in dealing with variable binding.

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 [8, 9] in her semantic normalization proof for the implication logic.

References

1. Altenkirch, T., Reus, B.: Monadic presentations of lambda terms using generalized inductive types. In: CSL, pp. 453–468 (1999)
2. Ambler, S.J., Crole, R.L., Momigliano, A.: A definitional approach to primitivexs recursion over higher order abstract syntax. In: MERLIN. ACM (2003)
3. Aydemir, B.E., Charguéraud, A., Pierce, B.C., Pollack, R., Weirich, S.: Engineering formal metatheory. In: POPL, pp. 3–15. ACM Press (2008)
4. Bellegarde, F., Hook, J.: Subsitution: A formal methods case study using monads and transformations. Sci. Comput. Program. **23**(2-3), 287–311 (1994)
5. 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)
6. Berger, U., Schwichtenberg, H.: An inverse of the evaluation functional for typed lambda-calculus. In: LICS, pp. 203–211. IEEE Computer Society (1991)
7. 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)
8. 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)
9. 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)
10. 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)
11. Geuvers, H.: Logics and Type Systems. Ph.D. thesis, University of Nijmegen (1993)
12. 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)
13. Harper, R., Licata, D.R.: Mechanizing metatheory in a logical framework. J. Funct. Program. **17**(4-5), 613–673 (2007)
14. 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)
15. 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)
16. 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)

17. Ilik, D., Lee, G., Herbelin, H.: Kripke models for classical logic. Ann. Pure Appl. Logic **161**(11), 1367–1378 (2010)
18. Kripke, S.: A Completeness Theorem in Modal Logic. J. Symb. Log. **24**(1), 1–14 (1959)
19. Kripke, S.: Semantical considerations on modal and intuitionistic logic. Acta Philos. Fennica **16**, 83–94 (1963)
20. Leroy, X.: A locally nameless solution to the poplmark challenge. Tech. Rep. 6098, INRIA (2007)
21. Mcbride, C., Mckinna, J.: Functional pearl: I am not a number: I am a free variable. In: In Haskell '04: Proceedings of the 2004 ACM SIGPLAN Workshop on Haskell, pp. 1–9. ACM Press (2004)
22. McKinna, J., Pollack, R.: Pure type systems formalized. In: TLCA, pp. 289–305 (1993)
23. McKinna, J., Pollack, R.: Some lambda calculus and type theory formalized. Journal of Automated Reasoning **23**(3-4), 373–409 (1999)
24. Mints, G.: Normal forms for sequent derivations. In: Kreiseliana, pp. 469–492. A K Peters, Wellesley, MA (1996)
25. Sato, M., Pollack, R.: External and internal syntax of the lambda-calculus. J. Symb. Comput. **45**(5), 598–616 (2010)
26. 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)
27. Urban, C.: Nominal techniques in isabelle/hol. J. Autom. Reasoning **40**(4), 327–356 (2008)