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

Formalising Logical Meta-theory

Semantical Normalisation using Kripke Models for Predicate Logic

Hugo Herbelin · Gyesik Lee

Received: date / Accepted: date

Abstract We present a new approach for dealing with binders when formalising a logical meta-theory. Our approach is similar to Pollack-McKinna's locally named approach with parameters for locally unbound variables and we introduce for this purpose the so-called notion of destined substitution. This approach has been used to formalise in Coq a proof of Kripke-based semantical cut-elimination for minimal first-order predicate logic. We also investigated how constants and free variables could be merged in such a formalisation.

Using parameters for locally unbound variables allows us to talk about well-formed terms and formulae without an extra syntax. Merging constants and free variables simplifies the process of formalisation in a significant way. The destined substitution opens a possibility of using simple structural induction where it is not accepted. We explain how to avoid the depth induction principle which is based on the complexity of formulae and seems to be essential when two sorts of variables are used for formalisation. Furthermore, we compare three equivalent ways of quantification: Cofinite, All-Fresh, and the traditional One-Fresh style. Finally, there is a meta-mathematical discussion on the meaning of free variables, bound variables, and constants.

We tried to make all the statements as acceptable as possible for logicians while keeping the computational contents and the conciseness of the semantical cut-elimination as simple as possible.

Keywords Formalisation with binders · locally named approach · normalisation by evaluation · intuitionistic predicate logic · Coq

Grants or other notes

For Gyesik Lee, this work was partially supported by the Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology(MEST) / Korea Science and Engineering Foundation(KOSEF), grant number R11-2008-007-01002-0.

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

G. Lee
ROSAEC Center, Seoul National University, Seoul, Korea
E-mail: gslee@ropas.snu.ac.kr

Mathematics Subject Classification (2000) 03B35 · 03F05 · 03F55

1 Introduction

Our main motivation is to investigate a new way of formal representation of logical formal systems with binders while trying not to depart too much from the traditional, conventional, standard way of informal arguments on paper using concrete names for variables. In particular, we deliberately renounced to use de Bruijn indices.

Using names to represent variables has a dark side with respect to term substitution: unrestricted substitution leads to variable capture. Usually one avoids this problem either by accepting substitution only when free variables are not captured or by assuming that this kind of substitution is always possible by renaming bound variables when necessary. So the definition of term substitution is not structurally recursive, but defined modulo some α-conversion. It is because one has to change some bound variables first in order to satisfy the Barendregt variable condition: the free variables of a term u should not be bound in another term t when u is substituted for a free variable x in t, cf. Barendregt [2].

There have been many suggestions in dealing with binders and substitution such as using de Bruijn indices rather than names [3, de Bruijn], locally nameless representation [7, Gordon] or, more recently, [1, Aydemir et al.], locally named representation [16, 17, McKinna and Pollack], higher-order abstract syntax [8, Harper et al.], etc. Section 2 gives a brief introduction to the well-known concrete approaches using some sorts of names.

Our proposal for dealing with variable binding (quantification) is based on McKinna-Pollack's locally named approach and uses dependent types to indicate the sets of free occurrences of bound variables: The set of terms and formulae depend on a list of variables. This allows us, above all, to talk about well-formed terms and formulae without an extra syntax. There are bright side and seamy side. We show how we maximise the bright side and how we overcome the seamy side. See Section 3.3.

Another difference from McKinna-Pollack's approach is that we do not consider terms which differ only by the name of bound variables to be equivalent. This may be surprising as it departs from the common usage of reasoning modulo α-conversion but in practice it is harmless because for any derivation that would consider some formulae modulo α-conversion, there is another one that differs from the original one only from the names of bound variables and that does not need α-conversion.

Another peculiarity of our work is that we only consider closed formulae in our derivations. Otherwise said, where two classes of names for bound and free variables are generally used, we here use constants (function symbols with arity 0) in place of free variables. This is consistent with the fact that free variables are not considered as a part of a given language. At first view, this makes a difference at the level of the model since the interpretation of constants is fixed in a given model while it is over the whole domain of interpretation for free variables. But since our soundness and completeness results are stated over *all* possible models, the interpretation of constants also range in practice over all their possible interpretations. The consequence is a significantly compacter formalisation because we save on substitution for free variables, hence no worry about variable capture. Furthermore, our formalisation of Normalisation by Evaluation (NbE) with respect to the intuitionistic predicate logic and Kripke semantics sheds light on some hidden roles of bound, free variables, and constants.

The third ingredient of our paper is the so-called *destined* substitution: Terms and formulae are parametrised by a superset of their exposed bound variables. The destined substitution opens a possibility of using simple structural induction where it is not allowed. We explain how to avoid the depth induction principle which is based on the complexity of formulae. The depth induction seems to be essential when two sorts of variables are used.

Furthermore, we compare three equivalent ways of quantification: Cofinite, All-Fresh, and the traditional One-Fresh style. Finally, there is a meta-mathematical discussion on the meaning of free variables, bound variables, and constants.

We use this approach for a formalisation of the soundness and completeness of an intuitionistic predicate theory with respect to Kripke semantics. For the presentation of the 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 just the absence of the cut rule). A disadvantage is that it is less "natural" than the so-called natural deduction, but such a structure has already been used in Coquand [4] and we found interesting to try an alternative approach.

We also adopt a Curry-Howard-de Bruijn style of proof system, i.e. we adopt a notion of proof whose associated cut-elimination process matches normalisation in λ-calculus (or, to be more precise, that matches normalisation in the $\overline{\lambda}$-calculus which is the variant of λ-calculus that fits the sequent calculus structure [9]).

This paper is organised as follows. In Section 2, we briefly introduce the well-known concrete approaches using some sorts of names. Section 3 explains new features of our approach based on locally named representation using dependent types. In Section 4 the so-called destined substitution and its impact are introduced. The sequent calculus and a comparison of three ways of formalising quantification are presented in Chapter 5 while Kripke semantics and its soundness and completeness with respect to the intuitionistic first-order predicate logic constitutes Section 6. This paper ends with some conclusions.

2 Representation of binders

We review a few ways binders can be represented when formalising a typing system.

2.1 Locally nameless representation and similar approaches

In locally nameless approach, cf. Aydemir et al. [1], bound variables are represented by de Bruijn indices while free variables are represented by names. The syntax of pure (pseudo-)λ-terms are defined as follows:

$$t ::= i \mid a \mid t\,t \mid \lambda\, t$$

where i varies over natural numbers and a over parameters, i.e., names for free variables. Definition of well-formed or well-typed terms needs more care about binders because of digit shifting. The substitution of a (locally) closed term s at position i is defined by:

$$\begin{array}{rcl} j[i\backslash s] & = & \text{if } j = i \text{ then } s \text{ else } j \\ a[i\backslash s] & = & a \\ (t_1\, t_2)[i\backslash s] & = & (t_1[i\backslash s])(t_2[i\backslash s]) \\ (\lambda\, t)[i\backslash s] & = & \lambda\, (t[i+1\backslash s]) \end{array}$$

The advantage of this approach lies in the combination of advantages of de Bruijn style and nominal approaches. Because of the syntactical distinction of free and bound variables, variable capture does not occur at all, hence substitution of locally closed terms can be defined by simple structural recursion. Furthermore, each term is unique in the sense that no two distinct terms are equivalent. The use of names for free variables makes lemmas and proofs more conventional.

Despite all the advantages, there remains still the main drawback of de Bruijn style approach. The above mentioned shifting of numbers is still necessary for definitions of deduction or typing rules. This makes some proofs, such as for substitution lemma, more tricky to handle.

2.2 Locally named representation

Similar to locally nameless approach, locally named representation uses two sorts of names for bound and free variables, respectively. This approach was first implemented in McKinna and Pollack [16,17] to formalise Pure Type System meta-theory, following the suggestion by Coquand [5]:

$$t ::= a \mid x \mid t\,t \mid \lambda x.t$$

where x varies over bound variables and a over free variables. There are two kinds of substitution, one for free variables and the other for bound variables, both defined by the structural induction on terms.

$$\begin{array}{rcl@{\qquad}rcl}
a[y\backslash s] & = & a & a[b\backslash s] & = & \begin{cases} s & \text{if } a = b, \\ a & \text{otherwise} \end{cases} \\
x[y\backslash s] & = & \begin{cases} s & \text{if } x = y, \\ x & \text{otherwise} \end{cases} & x[b\backslash s] & = & x \\
(t_1\,t_2)[y\backslash s] & = & (t_1[y\backslash s])(t_2[y\backslash s]) & (t_1\,t_2)[b\backslash s] & = & (t_1[b\backslash s])(t_2[b\backslash s]) \\
(\lambda x.t)[y\backslash s] & = & \begin{cases} \lambda x.\,t & \text{if } x = y, \\ \lambda x.\,t[y\backslash s] & \text{otherwise} \end{cases} & (\lambda x.t)[b\backslash s] & = & \lambda x.(t[b\backslash s])
\end{array}$$

In case of substitution for bound variables, variable capture could happen. But, in their work, renaming bound variables do not happen and substitution for a bound variable is used only in safe ways so as to avoid unintended variable capture.

2.3 Syntactic term uniqueness and beta reduction

Very recently, Sato and Pollack [20] proposed a way of avoiding α-conversion still using the locally named style. They defined the set $\mathbb{L}$ of canonical terms as follows:

$$\frac{}{X : \mathbb{L}} \qquad \frac{M : \mathbb{L} \quad N : \mathbb{L}}{(\mathtt{app}\, M\, N : \mathbb{L})} \qquad \frac{M : \mathbb{L} \quad x = H_X(M)}{(\mathtt{lam}[x][x/X]\, M) : \mathbb{L}} \;(*)$$

X varies over global variables (free variables) and x over local variables (bound variables) represented by natural numbers. H is a function finding the maximum number used as a binding variable in whose scope the (global) variable X is located: $H_X(M) = 0$

iff X is not used in the construction of M. And $H_X(M) = n+1$ iff X occurs in M and n is the largest binder among all the binders occurring in a path from the root of the term tree to an occurrence of X.

A main contribution of Sato-Pollack's paper is an alternative way of canonical representation of terms: Each term in $\mathbb{L}$ represents an α-equivalence class of terms. At the point of binding X in the term construction $\mathtt{lam}[x][X/x]\,M$, one chooses the height value $H_X(M)$ as the binder x. Moreover, $\mathbb{L}$ is closed under substitution.

On the other hand, $H_X(M) \neq H_X(N)$ in general when $M \to_\beta N$. That is, $\mathbb{L}$ is not syntactically closed under β-reduction which indicates that α-conversion is somehow necessary though.

Moreover, it raises the same question as in case of locally nameless whether it corresponds to informal practice because we have to write $\mathtt{lam}[H_X(M)]\,M$ instead of $\mathtt{lam}[x]\,M$. Actually, the mechanism seems to be very similar to the locally nameless style in Aydemir et al. [1] in the following two senses:

- The function H calculates the position of a global variable which corresponds to the role of de Bruijn indices.
- Because it is not syntactically closed under β-reduction, one has to struggle with number shifting as is the case for any approach using de Bruijn indices for binders. That is, it is not directly possible to develop a pure syntactic theory about β-reduction. One needs shifting of numbers which is a kind of α-conversion.

In general, it seems to be inevitable that any restriction to the definition of terms leads to this kind of problem with β-reduction.

3 Representation with locally traced names

Here we present a way of avoiding double definitions for well-formed terms/formulae by using dependent inductive types. Well-formed terms/formulae are formal expressions which could contain free occurrences of bound variables which are originally supposed to be bound.

In this section, we describe the formalisation of the first-order predicate logic with $\to$ and $\forall$ as the sole connectives using lists of unbound variables as parameters for dependent types.

3.1 Merger of free variables and constants

As for the formal representation of variables,

- we have two distinct classes of variables for bound and not-bound,
- we have a control on the set of exposed bound variables,
- we use names for both classes of variables.

One class of variables consists of bound variables, i.e., variables for binding and the other one is that of *not-bound* variables, usually reserved for free variables. We emphasise that we chose for using "not-bound" variables instead of using "free" variables. This is because we merge the role of free variables and that of constants.

The main consequence is a significantly simplified formalisation of the exactly same contents and results, about 1/4 less than the case with both free variables and constants. For instance, we need substitution only for bound variables because not-bound variables are considered in the syntactic level like constants although they take over the role of free variables. The idea is based on the following well-known facts in mathematical logic:

- During a syntactic proof, i.e. in a deduction tree, substitution of a term in a formula is only executed when a bound variable is involved. Even if we (can) formally introduce free variables, the substitution for free variables would not play any role in the whole process.
- Fresh constants are as good as fresh variables: if a formula derived from some assumptions contains a constant which does not occur in any of the assumptions, then we can replace the constant with arbitrary terms in the deduction tree. See Theorem 15 and Lemma 20.
- The completeness of the first-order predicate logic with respect to a semantics holds only for closed formulae, that is, formulae containing no free variables.

Our approach sheds also some light on the roles of free variables and constants from a *technically* new point of view: in the syntactic level, not-bound variables play the role of free variables as in (Cofin-$\forall_R$) of Figure 4 while, in the semantic level, they behave like real constants: their interpretation in a Kripke model is fixed as defined in Figure 5.

3.2 Dependent inductive types for pseudo-terms and -formulae

We assume that the language contains countably many constants (not-bound variables) and use natural numbers from the set `nat` of natural numbers to denote both bound and not-bound variables. `Bvar` stands for the denotation of bound variables while `Cst` for constants (not-bound variables).

Given a list of bound variables $m \in$ `list nat`, `pterm` m (resp. `pfml` m) denotes the set of pseudo-terms (resp. pseudo-formulae) in which only bound variables from m could occur free. Figure 1 describes the way of defining pseudo-terms and -formulae with lists of bound variables as type parameters. Intuitively, the parameters collects the free occurrence of bound variables. 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 free variables.

Notation: c, d, c_i, d_i vary over constants while x, y, x_i, y_i vary over bound variables. For simplicity, we assume two denumerably infinite, decidable sets[1] `predicate` of unary predicates and `function` of binary functions. f, g, f_i, g_i (resp. p, q, p_i, q_i) denote function (resp. predicate) symbols.

Place holders We use the notation *place holders* to call bound variables which occur free in a pseudo-term or -formula. We chose to use place holders for its syntactic meaning: A freely occurring bound variable in an expression has no meaning but waiting for being bound and substituted when needed. In this sense, it shares the basic idea of de Bruijn style numbering of bound variables, i.e., it remembers and holds a position in a formal expression.

[1] 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 $a = b \leftrightarrow f(a, b) = 0$.

Pseudo-terms:

$$\frac{x \in \texttt{nat} \quad (h : x \in m)}{\texttt{Bvar}\, x\, h \in \texttt{pterm}\, m} \qquad \frac{c \in \texttt{nat}}{\texttt{Cst}\, c \in \texttt{pterm}\, m} \qquad \frac{f \in \texttt{function} \quad t_1, t_2 \in \texttt{pterm}\, m}{\texttt{App}\, f\, t_1\, t_2 \in \texttt{pterm}\, m} \tag{1}$$

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

Pseudo-formulae:

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

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

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

Contexts: $\texttt{context} = \texttt{list formula}$

Occurrence of constants:

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

Occurrence of place holders:

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

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

Control of place holders The side condition $(h : x \in m)$ in the definition of $\texttt{Bvar}\, x\, h \in \texttt{pterm}\, m$ of (1) is one of the crucial points of the whole formalisation. In that way we control the information on place holders used in a term or a formula. Then $\texttt{pterm}\, m$ (resp. $\texttt{pfml}\, m$) collects all pseudo-terms (resp. pseudo-formulae) with possible place holders from the list m. The well-formed terms (resp. formulae) are then represented by the elements of $\texttt{pterm}\, nil$ (resp. $\texttt{pfml}\, nil$):

$$\texttt{term} := \texttt{pterm}\, nil \quad \& \quad \texttt{formula} = \texttt{pfml}\, nil$$

If e is a pseudo-term of -formula, let $\texttt{OC}(e)$ denote the set of constants occurring in e and $\texttt{PH}(e)$ the set of place holders occurring in e. Note that $\texttt{PH}(e)$ is already controlled by the parameter list:

Lemma 1 (Place holder property) *Let $e \in \{\texttt{term}\, m, \texttt{formula}\, m\}$. Then $\texttt{PH}(e) \subseteq m$. Therefore $\texttt{PH}(e)$ is an empty list when e is a well-formed term or a formula.*

For the formalisation, we use (finite) lists to denote finite sets of constants or place holders. In this paper, however, we use the usual set notations for a 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 m^{-x} for the removal of x from the list denoted by ℓ, the element relation $\in$ for the *being-in-a-list* relation.

The formal definition of *being-in-a-list* is a very important factor for our formalisation. If we use the following definition from the standard List library of Coq

$$x \in y :: m \text{ iff } (x = y \text{ or } x \in m) \tag{2}$$

then we cannot decide if two terms (resp. two formulae) are syntactically equal or not because of the parameter condition in the definition of pseudo-terms. Instead we need to deal only with decidable sets and use the following equivalent one:

$$x \in y :: m \quad \text{iff} \quad \text{if } x = y \text{ then } \texttt{True} \text{ else } x \in m \tag{3}$$

With help of the $\in$-relation, the subset relation can be used for the sublist relation: $m \subseteq \ell \quad \text{iff} \quad \forall v \in m\,(v \in \ell)$.

Thanks to the equivalence of (2) and (3) we can use all the existing libraries about lists without proving them extra. An advantage of using (3) is the uniqueness of proofs for being-in-a-list. Consequently the syntactic decidability holds:

Theorem 2 (Uniqueness of being-in-a-list) *Let A be a decidable set, $a \in A$, and $m \in \texttt{list}\, A$. Then*

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

Theorem 3 (Syntactic decidability) *Let m be a finite list of natural numbers.*

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

(Note that we assumed the decidability of the sets of function and predicate symbols.)

Theorem 3 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 purpose while de Bruijn for references are used. With our definition, the important parameter is the proof "h" which in fact is a canonical index (some "n^{th}" from the list m as explained in Theorem 1). The names are here only for decoration, with m telling in which order the names are numbered. Compared to the standard locally named representation which has no control of the exposed bound, in our case, we superimpose de Bruijn and names so that α-equivalence (implicit in Theorem 3) is easily checked.

3.3 Dependent types with list parameters

Using two sorts of variables for bound and not-bound variables has an unpleasant feature: not all syntactic expressions are meaningful. Indeed this is always the case where two different sorts of variables are used. It is therefore usually required to give an extra syntax for the definition of well-formed expressions for the adequacy of formalisation. In [16, 17] the set of well-formed terms are called `Vclosed`, and in [1] an extra typing rules for *locally closed* terms are given.

In our approach, pseudo-terms/-formulae are classified into dependent inductive types $\texttt{pterm}\, m/\texttt{pfml}\, m$ depending on the type parameter m. Defining pseudo-terms and -formulae as type families indicates some technical issues. Below are some seamy and bright aspects of using type parameters.

Bright side and benefits

1. It enables us to deal with well-formed terms and formulae in a compact way because there is no need for an additive definition of well-formed expressions.
2. We have an easy control of bound variables occurring free in a pseudo-term/-formula. If e.g. $t \in \texttt{pterm}\, m$, then m includes every occurrence of bound variables occurring free in t. In many situations this helps us simplify some formal definitions and proofs. For example, the soundness and completeness of Kripke semantics with respect to the intuitionistic predicate logic. Cf Theorem 21 and Theorem 25.
3. There are also reasons in the meta-level why a compact handling of well-formed expressions are important. First, only well-formed terms and formulae have a meaning and correspond to ordinary defined 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 defined only for well-formed formulae. Cf. Figure 4.

Seamy side and solutions

1. Proof of the decidability of the classes $\texttt{pfml}\, m$ is a little delicate[2].

$$\forall (A, B : \texttt{pfml}\, m)\, (A = B \;\vee\; A \neq B)$$

   To prove this, one needs to work with dependent equality for which there is a standard library for Coq such as `Eqdep_dec`.
2. From the point of term structure, a pseudo-term or a -formula belongs to infinitely many classes: a term t e.g. from $\texttt{pterm}\, m$ belongs also to $\texttt{pterm}\, k$ for all k including m as a sublist.
   However, for any term t in the style of McKinna-Pollack's locally named representation, there is the canonical and smallest list m such that $t \in \texttt{pterm}\, m$, i.e., the collection of all bound variables occurring free in t.
   Furthermore, the relationship between two classes can be connected by homomorphic *lifting* functions from $\texttt{pterm}\, m$ to $\texttt{pterm}\, k$. The lifting functions are homomorphic in the sense that they preserve syntactic and semantic properties such as syntactic decidability, substitution, and validity in a Kripke model, etc. See Section 3.4, Lemma 10, and Theorem 17.
3. Definition of substitution requires a special care. Intuitively, one would like to have the following form:

$$\begin{gathered} A \in \texttt{pfml}\,(x :: m) \;\;\&\;\; u \in \texttt{term} \quad\Rightarrow\quad A[x\backslash u] \in \texttt{pfml}\, m \\ \text{or} \\ A \in \texttt{pfml}\, m \;\;\&\;\; u \in \texttt{term} \quad\Rightarrow\quad A[x\backslash u] \in \texttt{pfml}\, m^{-x} \end{gathered}$$

   However, this would make the definition of substitution complicated because one should struggle with the formula depth. This complexity would lead to many technical difficulties during the formalisation. We solved this problem by introducing the so-called *destined substitution*. See Section 4.

Lifting: Let $t \in \texttt{pterm}\, m$, $\texttt{PH}(t) \subseteq \ell$. Then $\texttt{tlift}^{m,\ell} : \texttt{pterm}\, m \to \texttt{pterm}\, \ell$ is recursively defined: (We write $\texttt{tlift}$ for $\texttt{tlift}^{m,\ell}$.)

$$\begin{aligned}
\texttt{tlift}(\texttt{Bvar}\, x\, h) &= \texttt{Bvar}\, x\, h' \\
\texttt{tlift}(\texttt{Cst}\, c) &= \texttt{Cst}\, c \\
\texttt{tlift}(\texttt{App}\, f\, t_1\, t_2) &= \texttt{App}\,(\texttt{tlift}(t_1))\,(\texttt{tlift}(t_2))
\end{aligned} \tag{4}$$

where h' is a canonical proof that $x \in \ell$ obtained from the assumption $\texttt{PH}(t) \subseteq \ell$.

Let $A \in \texttt{pfml}\, m$, $\texttt{PH}(A) \subseteq \ell$. Then $\texttt{flift}^{m,\ell} : \texttt{pfml}\, m \to \texttt{pfml}\, \ell$ is recursively defined: (We write $\texttt{flift}$ for $\texttt{flift}^{m,\ell}$.)

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

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

Fig. 2 Lifting

3.4 Lifting of parameters

Intuitively, $\texttt{term}$ is a subset of $\texttt{pterm}\, m$ for any $m \in \texttt{list nat}$, that is, there are infinitely many possibility of representing a (pseudo-)term differing not only in the names of binders, but also in the list parameter: even if two pseudo terms are α-equivalent in the classical sense, they could be syntactically different because the conditional part of being-in-a-list in the definition of a pseudo-term varies for each list parameter.

However, we will see that this point can be just ignored. The conditional part of being-in-a-list is inessential so long as the list parameter contains all the necessary place holders needed for the construction of an expression. The point is to respect the fact that only the bound variables which really occur count, see Figure 2.

In the definition of $\texttt{tlift}^{m,\ell}$ (resp. $\texttt{flift}^{m,\ell}$) we demand $\texttt{PH}(t) \subseteq \ell$ (or $\texttt{PH}(A) \subseteq \ell$) instead of $m \subseteq \ell$. Note furthermore that $\texttt{tlift}^{m,\ell}(t)$ does not change anything in t except for the proof part of being-in-a-list and that the choice of h' in (4) is not important at all because of the proof-uniqueness shown in Theorem 2. This indicates also that several applications of lifting is the same as one application and that the syntactic equality between two expressions is preserved.

Given an expression $e \in \{\texttt{pterm}\, m, \texttt{pfml}\, m\}$, we write $\texttt{lift}$ for $\texttt{tlift}$ or $\texttt{flift}$ depending on e. Then the following two lemmas can be proved inductively.

Lemma 4 (Idempotence) *Let $e \in \{\texttt{pterm}\, m, \texttt{pfml}\, m\}$ and $m \subseteq \ell \subseteq k$. Then*

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

Lemma 5 (Equality preservation) *Let $t, s \in \texttt{pterm}\, m$ (resp. $A, B \in \texttt{pfml}\, m$) and $\texttt{PH}(t), \texttt{PH}(s) \subseteq \ell$ (resp. $\texttt{PH}(A), \texttt{PH}(B) \subseteq \ell$). Then*

1. *If $t = s$, then $\texttt{tlift}(t) = \texttt{tlift}(s)$.*
2. *If $A = B$, then $\texttt{flift}(A) = \texttt{flift}(B)$.*

[2] With respect to Coq.

Remark 6 *Lifting is originally invented to overcome the syntactic difficulties caused by the list parameters. For example, substitution of terms is not possible without using lifting. However, there is also a hidden meta-theoretic significance: it guarantees the well-formedness of a resulting pseudo-term after all the place holders are replaced with well-formed terms. See Remark 8 for a more concrete explanation.*

3.5 Lists as environments

In our formal presentation all sorts of environments are realised by lists: parameters of place holders, term lists for simultaneous substitution, formula lists for contexts, and even associations needed for interpretation of pseudo-terms in a Kripke model, see Figure 5. We emphasise that, in dealing with list-style environments, the first occurrence from the left is most important because the interpretation depends exactly on the first occurrence of appropriate expressions. We only need to add an element as the new head of a list when we want to get an intended interpretation:

> If $f : A \to B$ is a finite function represented by a list and if we want to define a new function g such that $g(x) = z$ and $g(y) = f(y)$ if $x \neq y$, then g can be represented by $(x, z) :: f$. Therefore, no need for "freshness" condition when expanding environments and no "lookup"-like functions are necessary.

4 Destined substitution and structural induction

Simultaneous substitution of finitely many terms for place holders in a term or a formula is recursively defined based on the term-/formula-structure. As demonstrated in Stoughton [21], it is more efficient for our formalisation to define simultaneous substitution than to define substitution of one single term.[3] Only well-formed terms are substituted because substituting pseudo-terms could cause variable capture. In this way, we have avoided α-conversion during the whole formalisation.

Remark 7 *Note that this problem of possible variable capture when arbitrary pseudo-terms are allowed to be substituted is not restricted to the locally named approach. It is a general problem when using two sorts of variables.*

For simultaneous substitution, associations of the form $\eta = (x_1, u_1), ..., (x_n, u_n)$ are used where $x_i \in \texttt{nat}$ and $u_i \in \texttt{term}$. Set $\mathsf{dom}(\eta) := \{x_1, ..., x_n\}$. For $y \in \mathsf{dom}(\eta)$, we write

$$\eta(y) := u_i \quad \text{where } i = \min\{j : y = x_j\} \tag{5}$$

4.1 Destined substitution

In the definition of substitution, we use a finite list ℓ of natural numbers as an extra argument. Intuitively, ℓ is a superset of the exposed bound variables after the substitution, i.e., collecting bound variables which are really bound by the quantification for

[3] "If my recursion is not structural, I am using the wrong structure", McBride [15, p. 241].

Simultaneous substitution: Let $m, \ell \in \texttt{list}\,\texttt{nat}$, $t \in \texttt{pterm}\,m$, $A \in \texttt{pfml}\,m$ and $\eta = (x_1, u_1), ..., (x_n, u_n)$ where $x_j \in \texttt{nat}$ and $u_j \in \texttt{term}$ for all $j \in \{1, ..., n\}$.

Recursive definition of $t[.\backslash\eta]_\ell \in \texttt{pterm}\,\ell$:

$$(\texttt{Bvar}\,y\,h)[.\backslash\eta]_\ell = \begin{cases} \texttt{Bvar}\,y\,h' & \text{if } y \in \ell \\ \texttt{tlift}\,u_j\,(\texttt{nil\_term}\,u_j\,\ell) & \text{if } y \notin \ell \text{ and } j = \min\{i : y = x_i\} \\ \texttt{Cst}\,0 & \text{otherwise} \end{cases} \tag{6}$$

$$(\texttt{Cst}\,c)[.\backslash\eta]_\ell = \texttt{Cst}\,c$$

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

where

- h' is the proof of $y \in \ell$ given by the assumption,
- $\texttt{nil\_term}$ is a function which, given a term $u \in \texttt{term}$ and a list ℓ, provides a proof that $\texttt{PH}(u) = nil \subseteq \ell$.

Recursive definition of $A[.\backslash\eta]_\ell \in \texttt{pfml}\,\ell$:

$$(P\,t)[.\backslash\eta]_\ell = P\,(t[.\backslash\eta]_\ell)$$

$$(A \to B)[.\backslash\eta]_\ell = A[.\backslash\eta]_\ell \to B[.\backslash\eta]_\ell$$

$$(\forall x\,B)[.\backslash\eta]_\ell = \forall x\,(B[.\backslash\eta]_{x::\ell}) \tag{7}$$

Fig. 3 Destined substitution

which no substitution can be performed: The extension of the argument ℓ by x in (7) says that any substitution for x is forbidden while the third case in (6) says that the uninteresting place holders are abandoned.

In this way we achieve $t[.\backslash\eta]_\ell \in \texttt{pterm}\,\ell$ and $A[.\backslash\eta]_\ell \in \texttt{pfml}\,\ell$ depending only on ℓ and independent of t, A or η. In particular, we have $t[.\backslash\eta]_{nil} \in \texttt{term}$ and $A[.\backslash\eta]_{nil} \in \texttt{formula}$.

Another, more intuitive way of defining simultaneous substitution would be as follows: (only the critical cases are considered)

$$(\texttt{Bvar}\,y\,h)[.\backslash\eta] = \begin{cases} \texttt{Bvar}\,y\,h & \text{if } y \notin \mathsf{dom}(\eta) \\ \texttt{tlift}\,u_j\,(\texttt{nil\_term}\,u_j\,m) & \text{if } u_j = \eta(y) \end{cases}$$

$$(\forall x\,B)[.\backslash\eta] = \forall x\,(B[.\backslash\eta^{-x}]) \tag{8}$$

where η^{-x} is the association obtained from η by deleting all the occurrence of (x, u) for some term u. However, there are at least two negative features:

1. Every time we perform substitution or want to prove some properties about substitution, we need to deal with $(\cdot)^{-x}$ function.
2. A more serious problem is caused by the list parameter. When we define substitution of η in $t \in \texttt{pterm}\,m$, we would like to have that the resulting pseudo-term $t[.\backslash\eta]$ belongs to $\texttt{pterm}\,(m\backslash\mathsf{dom}(\eta))$. However, a simple recursive definition of substitution is not possible because we have to heavily struggle with lifting of list parameters, hence with formula complexity. This would make the whole development messy and intricate.

Remark 8 (Meta-theoretic meaning of the parameter ℓ) *Although it is syntactically complicated to get an appropriate definition of substitution in the style of* (8), *one can easily see that* $\mathtt{PH}(A[.\backslash\eta]) \subseteq m\backslash \mathsf{dom}(\eta)$, *so*

$$\mathtt{flift}^{m,m\backslash \mathsf{dom}(\eta)}(A[.\backslash\eta]) \in \mathtt{pterm}\,(m\backslash \mathsf{dom}(\eta)).$$

If we suppose furthermore that $m = \mathsf{dom}(\eta)$, *then* $\mathtt{PH}(A[.\backslash\eta] = nil$ *is provable. That is,* $\mathtt{flift}(A[.\backslash\eta]) \in \mathtt{formula}$. *All this indicates that* `flift` *(resp.* `tlift`*) brings with itself that* $A[.\backslash\eta]$ *is well defined. In our definition in Figure 3, the parameter* ℓ *takes over this role of lifting and has minimalised the use of lifting.*

The substitution of a term for a place holder is a special case of a simultaneous substitution: Let $e \in \{\mathtt{pterm}\,m, \mathtt{pfml}\,m\}$

$$e[x\backslash u]_\ell = e\,[.\backslash((x,u) :: nil)]_\ell$$

The following lemmas play the key role in the proof of universal completeness of Kripke semantics with respect to the intuitioinistic predicate logic, cf. Theorem 25.

Lemma 9 (Substitution Lemma) *Let* $e \in \{\mathtt{pterm}\,m, \mathtt{pfml}\,m\}$ *and* $u \in \mathtt{term}$ *and* $\eta \in \mathtt{list}\,(\mathtt{nat} * \mathtt{term})$.

$$(e[.\backslash\eta]_{y::\ell})[y\backslash u]_\ell = e[.\backslash(y,u) :: \eta]_\ell$$

Lemma 10 (Substitution and lifting) *Let* $e \in \{\mathtt{pterm}\,m, \mathtt{pfml}\,m\}$, $\mathtt{PH}(e) \subseteq k$, $\eta \in \mathtt{list}\,(\mathtt{nat} * \mathtt{term})$, *and* $\ell \in \mathtt{list}\,\mathtt{nat}$.

$$(\mathtt{lift}^{m,k}(e))\,[.\backslash\eta]_\ell = e\,[.\backslash\eta]_\ell$$

That is, lifting has no effect on substitution.

A more practical aspect of using an extra parameter ℓ is explained in the following section where structural induction and depth induction are compared.

4.2 Structural vs. depth induction

A main consequence of working with two sorts of variables is that an unintended variable capture during a substitution can be avoided. Another important one, this time less delightful, is that we sometimes cannot use structural induction principle anymore to prove properties on well-formed expressions with binders. There are cases where we have to work with induction on the complexity of the syntactic expressions, i.e., a well-founded induction on the formula depth.

Depth induction principle Given a property $\mathcal{P}$ on (well-formed) formulae, we usually use the following induction principle to prove that the property holds for all well-formed formulae:

$$\frac{\begin{array}{c}\forall(P \in \texttt{predicate})(t \in \texttt{term})\ [\,\mathcal{P}(P\,t)\,]\\ \forall(A, B \in \texttt{formula})\ [\,\mathcal{P}(A) \Rightarrow \mathcal{P}(B) \Rightarrow \mathcal{P}(A \to B)\,]\\ \forall(A \in \texttt{pfml}\,(x :: nil))\ [\,\mathcal{P}(A(c)) \Rightarrow \mathcal{P}(\forall x\, A(x))\,]\ \text{ for some fresh constant } c\end{array}}{\forall(A \in \texttt{formula})\ [\,\mathcal{P}(A)\,]} \tag{9}$$

where $A(c) := A(x)[x\backslash\texttt{Cst}\,c]$.

However, (9) is not exactly a structural induction, but an induction on the complexity of formulae: $A(c)$ is not a syntactic subformula of $\forall x\, A(x)$. Indeed, we need something measuring the complexity of formulae:

$$\begin{aligned}\texttt{depth}(P\,t) &= 0\\ \texttt{depth}(A \to B) &= \max(\texttt{depth}(A), \texttt{depth}(B)) + 1\\ \texttt{depth}(\forall x\, A) &= \texttt{depth}(A) + 1\end{aligned}$$

Then we can use the induction on $\texttt{depth}(A)$ to prove a property for all formulae since $\texttt{depth}(A(c)) = \texttt{depth}(A(x)) < \texttt{depth}(\forall x\, A)$.

Definition 11 (Depth Induction principle)

$$\forall(n \in \texttt{nat})(A \in \texttt{formula})\ [\,\texttt{depth}(A) \le n \to \mathcal{P}(A)\,]$$

The case $n = 0$ deals only with atomic formulae since otherwise the depth of a formula is larger than 0. If $n = m + 1$, then complexer formulae can be considered using a subsidiary induction on the structure of A like (9). Now the original claim $\forall(A \in \texttt{formula})\,\mathcal{P}(A)$ follows easily by putting $n := \texttt{depth}(A)$.

Remark 12 *The induction principle* (9) *corresponds exactly to the well-founded induction on* `length` *in McKinna-Pollack [17] and to the strengthened induction principle for* $\mathbb{L}$*, the set of canonical* λ*-terms, in Sato and Pollack [20].*

Induction with destined substitution The depth induction is a universal tool in *proving* mathematical properties of all formulae. From the computational point of view, however, the depth induction is a bit less efficient than the structural induction: the depth induction uses two inductions, one for the depth and the other for the structure of a formula. Here we present a way of avoiding the depth induction by using the destined substitution.

Definition 13 (Induction with destined substitution)

$$\frac{\begin{array}{c}\forall(P \in \texttt{predicate})(t \in \texttt{pterm}\,m)\ [\,\mathcal{P}(P\,(t[.\backslash\eta]_{nil}))\,]\\ \forall(A, B \in \texttt{pfml}\,m)\ [\,\mathcal{P}(A[.\backslash\eta]_{nil}) \Rightarrow \mathcal{P}(B[.\backslash\eta]_{nil}) \Rightarrow \mathcal{P}((A \to B)[.\backslash\eta]_{nil})\,]\\ \forall(A \in \texttt{pfml}\,(x :: m))\exists(c \notin \texttt{OC}(A))\ [\,\mathcal{P}(A[.\backslash(x, \texttt{Cst}\,c) :: \eta]_{nil}) \Rightarrow \mathcal{P}(\forall x\, A(x)[.\backslash\eta]_{nil})\,]\end{array}}{\forall(A \in \texttt{pfml}\,m)\,\mathcal{P}(A[.\backslash\eta]_{nil})}$$

The trick here is based on the idea that $A[.\backslash\eta]_{nil} \in \texttt{formula}$ although $A \in \texttt{pfml}\,m$ is an arbitrary pseudo-formula. One applies then the structural induction on A because A can be considered syntactically separated from other arguments for the simultaneous substitution.

We apply this induction principle for the proof of universal completeness of Kripke semantics with respect to the intuitionistic logic LJT, cf. Theorem 25.

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

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

$$\frac{\Gamma \mid A(x)[x\backslash t] \vdash C}{\Gamma \mid \forall x\, A(x) \vdash C}\ (\forall_L) \qquad \frac{\forall (c \notin L)\ [\, \Gamma \vdash A(x)[x\backslash c]\,]}{\Gamma \vdash \forall x\, A(x)}\ (\mathit{Cofin}\text{-}\forall_R)$$

where $L \in$ `list nat` in the rule $(\mathit{Cofin}\text{-}\forall_R)$.

Fig. 4 Cut-free LJT

5 Intuitionistic sequent calculus LJT and quantification

For the presentation of the 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 just the absence of the cut rule). We also adopt a Curry-Howard-de Bruijn style of proof system, i.e. we adopt a notion of proof whose associated cut-elimination process matches normalisation in λ-calculus (or, to be more precise, that matches normalisation in the $\overline{\lambda}$-calculus which is the variant of λ-calculus proper to the sequent calculus structure [9]).

The Curry-Howard-de Bruijn approach requires to be able to distinguish between the different occurrences of a given formula in the context Γ of a sequent $\Gamma \vdash A$. There are typically two canonical ways to achieve this: either one considers contexts as sets of *named* formulae where the names are used to distinguish between the different occurrences of the same formula, or we consider contexts as list (i.e. ordered sets) of formulae in which case, the underlying order provides the way to distinguish between the different occurrences of the same formula. This latter approach is the one we adopted (cf. Section 3.5)[4].

5.1 Intuitionisitic sequent calculus LJT

The Gentzen-style sequent calculus LJT is obtained from the intuitionistic sequent calculus LJ by restricting the use of the left introduction rules of the implication and the universal quantification. In that way, we attain a one-to-one correspondence between cut-free proofs in LJT and normal λ-terms, cf. Herbelin [9,10]. Obviously, LJT is complete for intuitionistic reasoning.

Figure 4 describes the rules of cut-free LJT. In cut-free LJT, a sequent has one of the forms $\Gamma \mid A \vdash C$ or $\Gamma \vdash C$, where the context $\Gamma = \{A_1, ..., A_n\}$ is a list of well-

[4] 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 a same formula (e.g. Troelstra and Van Dalen's crude discharge convention [22, Ch. 1]), this amounts to giving names to formulae (see Geuvers [6] for a discussion).

formed formulae. The right side of "|" in the antecedence is called *stoup*. $\Gamma, \Gamma', \Delta, ...$ will vary over contexts.

The weakening lemma can be easily proved by a simple induction on the derivation.

Lemma 14 (Weakening) *Let A, C be formulae and Γ, Γ' contexts such that $\Gamma \subseteq \Gamma'$.*

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

Choice of a fresh constant Given a context Γ, choosing a fresh constant $c \notin \mathtt{OC}(\Gamma) := \bigcup_{A \in \Gamma} \mathtt{OC}(A)$ is required sometimes in reasoning about quantification such as in proving soundness and completeness. For the choice function we adopted the idea given by O'Connor [18]. Given a context Γ, a fresh constant can be chosen as follows:

$$\mathtt{fresh\_out}(\Gamma) := \max\{a \in \mathtt{nat} : a \in \mathtt{OC}(\Gamma)\} + 1 \tag{10}$$

5.2 Universal quantification, a variable binding

The way of defining an adequate quantification rule deserves a special attention. Indeed, quantification is one of the key points besides the difficulties connected to substitution and variable capture. In particular, we have

- to formalise a proof system with an adequate treatment of the freshness condition in the $\forall$ right introduction rule;
- to state and prove a weakening lemma for this proof system that preserves the freshness condition of derivations;
- to define a notion of substitution of the variables occurring below a binder (this is needed e.g. to define the notion of forcing $\omega \Vdash A$ as the case (13) in Figure 5.);
- to characterise a subset of terms that will serve as standard model for the completeness proof (we have then to ensure that any variable used in a binder avoids the variables in terms).

For our formalisation we chose for cofinite quantification style for the $\forall$ right introduction based on Aydemir et al. [1], see (*Cofin*-$\forall_R$). The idea is to consider all but those in some finite set collecting possible constants (not-bound variables) already used in the derivation.

In this section, we compare three approaches to formalisation of quantification: cofinite quantification, one-fresh quantification, and all-fresh-quantification.

Cofinite quantification The first ideas about cofinite quantification can be found in Pitts [19] and Krivine [14]. Aydemir et al. [1] demonstrates how effectively a cofinite quantification can be used for the formalisation of a system with binders. In particular, using cofinite quantification one can avoid the need for reasoning about equivariance (the fact that fresh names can be renamed in derivations). We confirm that it is really an efficient tool for dealing with binders: compact syntax, strong enough as an introduction rule, and with a strengthened induction principle because of a flexible way of dealing with fresh variables, constants in our case.

Traditional One-Fresh quantification In mathematical logic, one usually says that if $\Gamma \vdash A(c)$ is derivable for a fresh (not-bound) variable (in our formalisation, a fresh constant) c which does not occur in A, Γ, then $\Gamma \vdash \forall x\, A(x)$ holds.

$$\frac{\Gamma \vdash A(x)[x\backslash \texttt{Cst}\, c] \quad c \textit{ fresh}}{\Gamma \vdash \forall x\, A(x)} \; (\textit{OneFresh-}\forall_R)$$

Indeed, the rule ($\textit{OneFresh-}\forall_R$) reflects the following intuition: if the premise holds for *some* c which does not occur free in Γ nor in $\forall x\, A(x)$, c should not have been affected by any operation during the deduction, and therefore it should be possible for arbitrary term t to arrive at $\Gamma \vdash A(t)$ without any modification of the proof.

However, this ($\textit{OneFresh-}\forall_R$) turns out to be too weak to directly provide the expected induction principle: the premise holds for the specific variable/constant c only and if we need to use it with another fresh variable/constant, the induction hypothesis is not applicable.

Let LJT_o be LJT with ($\textit{OneFresh-}\forall_R$) instead of ($\textit{Cofin-}\forall_R$). Although LJT_o enjoys the weakening lemma, it cannot be proved by a simple induction on the derivation. The critical case is ($\textit{OneFresh-}\forall_R$): even if c is fresh in Γ, there is no guarantee that it is fresh in Γ'. The proof demands e.g. that LJT_o is closed under (*renaming*):

$$\frac{\Gamma \vdash A(x)[x\backslash c] \quad c, d \notin \texttt{OC}(\forall x\, A(x) :: \Gamma)}{\Gamma \vdash A(x)[x\backslash d]} \; (\textit{renaming})$$

There are two ways to address this issue: either use an induction on the *length* of derivation, or generalise the induction hypothesis into

$$\Gamma \vdash A \text{ and } \Gamma \subseteq \Gamma' \text{ implies that for all renaming of variables } \sigma \text{ holds } \Gamma'[\sigma] \vdash A[\sigma]$$

All-Fresh quantification As explained above, the ($\textit{OneFresh-}\forall_R$) rule causes a bit too much computational cost than expected because we need to use (*renaming*) extra in many situations. McKinna and Pollack [16,17] strengthens the induction hypothesis by demanding that all possible fresh variable should be considered:

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

Let LJT_a be LJT where ($\textit{Cofin-}\forall_R$) is replaced with ($\textit{AllFresh-}\forall_R$). Then (*weakening*) for LJT_a can be proved easily by a simple induction on the derivation without (*renaming*). In fact, (*renaming*) is not used anywhere even when we worked with LJT_a instead of LJT. (*renaming*) would be necessary only when we want to prove the equivalence of LJT_a and LJT_o. A proof of (*renaming*) in LJT_a would also require an induction on the derivation length.

Equivalence One can prove the equivalence of the three ways of quantification based on the following two facts about the substitution lemma which indicates that a fresh constant is as good as a fresh variable:

Theorem 15 (Fresh constants, renaming) *Assume c is a constant which does not occur in Γ. Then for an arbitrary term t the following hold:*

1. *If $\Gamma \vdash A$, then $\Gamma \vdash A[c\backslash t]$.*
2. *The derivation length of $\Gamma \vdash A[c\backslash t]$ is the same as that of $\Gamma \vdash A$.*

Now we can prove the equivalence of the three systems. Let $\vdash_{\overline{a}}$ denote the derivation in LJT_a and $\vdash_{\overline{o}}$ in LJT_o.

Theorem 16 LJT, LJT_a *and* LJT_o *are equivalent.*

Proof We prove the equivalence in the following way.

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

– $\Gamma \vdash_{\overline{o}} A \Rightarrow \Gamma \vdash A$: The proof can be done by the induction on the derivation. The critical case is the $\forall$ right introduction rule. Assume $\Gamma \vdash_{\overline{o}} A(d)$ for some fresh constant. Then $\Gamma \vdash A(d)$ by the induction hypothesis. Using (*renaming*) we also have $\Gamma \vdash A(c)$ for all fresh constant c. Now put $L := \mathtt{OC}(A) \cup \mathtt{OC}(\Gamma)$, then the premise of (*Cofin*-$\forall_R$) is satisfied.
– $\Gamma \vdash A \Rightarrow \Gamma \vdash_{\overline{a}} A$: The proof can be done by the induction on the derivation length of $\Gamma \vdash A$. The critical case is the $\forall$ right introduction rule. Assume $\Gamma \vdash A(c)$ for all $c \notin L$, L a finite set of constants. We need to show $\Gamma \vdash_{\overline{a}} A(d)$ for all $d \notin \mathtt{OC}(A) \cup FV(\Gamma)$. If $d \notin L$, then we are done with the induction hypothesis. Assume now $d \in L$. Since there are infinitely many constants, we can find a fresh one d' such that $d' \notin L \cup \mathtt{OC}(A) \cup \mathtt{OC}(\Gamma)$. Then $\Gamma \vdash A(d')$, hence by (*renaming*) $\Gamma \vdash A(d)$ with the same derivation length as $\Gamma \vdash A(d')$. Therefore we can apply the induction hypothesis on $\Gamma \vdash A(d)$ to get $\Gamma \vdash_{\overline{a}} A(d)$.
– $\Gamma \vdash_{\overline{a}} A \Rightarrow \Gamma \vdash_{\overline{o}} A$: Trivial.

6 Kripke semantics, soundness and completeness

Kripke semantics was created in the late 1950s and early 1960s by Saul Kripke [12,13]. It was first made for modal logic, and later adapted to intuitionistic logic and other non-classical systems.

6.1 Kripke semantics

Syntactically, a Kripke model can be formalised as a record type with 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 sentences in the extended language with the new constant symbols for each domain element, cf. Figure 5:

– We use finite associations for the interpretation of place holders. However, the interpretation of pseudo-terms is made total by ignoring dummy place holders which are not supposed to get an interpretation.
– In the case of universal quantification of the forcing definition, $(y, d) :: \eta$ is used to denote the association η' obtained form η such that $\eta'(y) = d$.
– The forcing definition at the universal quantification case (13) is much simpler than the standard definition, where the domain depends on worlds:

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

Indeed, they are "functionally equivalent" in the sense that soundness and completeness hold in both cases. Cf. Herbelin and Lee [11].

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 prime sentences in the language extended with constant symbols for each element of $\mathcal{D}$ such that

$$(w \leq w' \quad \& \quad w \Vdash P\,d) \quad \Rightarrow \quad w' \Vdash P\,d$$

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

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

$$(\mathsf{Bvar}\,x\,h)[\eta] = \begin{cases} \eta(x) & \text{if } x \in \mathsf{dom}(\eta) \\ V(0) & \text{otherwise} \end{cases} \tag{11}$$

$$(\mathsf{Cst}\,c)[\eta] = V(c) \tag{12}$$

$$(f\,t_1\,t_2)[\eta] = V(f)(t_1[\eta], t_2[\eta])$$

Here $\eta(x)$ is similarly defined as in (5).

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

$$\begin{aligned} w \Vdash (P\,t)[\eta] &\text{ iff } w \Vdash P\,(t[\eta]) \\ w \Vdash (A \to B)[\eta] &\text{ iff for all } w' \geq w,\ w' \Vdash A[\eta] \text{ implies } w' \Vdash B[\eta] \\ w \Vdash (\forall x\,A)[\eta] &\text{ iff for all } d \in \mathcal{D},\ w \Vdash A[(y, d) :: \eta] \end{aligned} \tag{13}$$

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

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

Fig. 5 Kripke semantics

– The proof parameter of a term is simply ignored during the interpretation, see (11). This implies that lifting of place holders does not have any impact on the Kripke semantics. Cf. Theorem 17.

Theorem 17 (Lifting-irrelevance) *Given a Kripke model $\mathcal{K} = (\mathcal{W}, \leq, \Vdash, \mathcal{D}, V)$, $w \in \mathcal{W}$, $\eta \in \mathtt{list}\,(\mathtt{nat} * \mathcal{D})$, and $t \in \mathtt{pterm}\,m$, $A \in \mathtt{pfml}\,m$, we have*

$$\begin{aligned} t[\eta] \quad &= \quad \mathtt{tlift}(t)[\eta] \\ w \Vdash A[\eta] \quad &\Longleftrightarrow \quad w \Vdash \mathtt{flift}(A)[\eta] \end{aligned}$$

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

Lemma 18 (Monotonicity) *Given a Kripke model $\mathcal{K} = (\mathcal{W}, \leq, \Vdash, \mathcal{D}, V)$, $w \leq w' \in \mathcal{W}$, $\eta \in \mathtt{list}\,(\mathtt{nat} * \mathcal{D})$, and $A \in \mathtt{pfml}\,m$, we have*

$$w \Vdash A[\eta] \Rightarrow w' \Vdash A[\eta]$$

6.2 Soundness and fresh constants

For the proof of soundness of the LJT with respect to the Kripke semantics, the case (*Cofin*-$\forall_R$) requires a special care:

$$\frac{(\forall c \notin L)\ [\,\Gamma \vdash A(x)[x\backslash \mathtt{Cst}\, c]\,]}{\Gamma \vdash \forall x\, A(x)}\ (\mathit{Cofin}\text{-}\forall_R) \tag{14}$$

This is because, given a Kripke model, each constant is associated with a fixed value from the domain, cf. (12), while the interpretation of the universal quantification involves all possible values from the domain. Therefore, at a first glance, the premise of (Cofin-$\forall_R$) seems to provide too weak an induction hypothesis. This kind of problem would not occur if we have introduced free variables and defined the deduction rule in a standard way, cf. [11].

The solution lies again in the fact that fresh constants are as good as fresh variables. Syntactically, this is already demonstrated in Lemma 15. In the semantic level, This corresponds to creating a new Kripke model from a given one such that their forcing relations behave almost the same.

Definition 19 *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. The following lemma is obviously true since the evaluation of a constant which does not occur has no influence on the forcing relation.

Lemma 20 (Forcing with fresh constants) *Given a Kripke model $\mathcal{K} = (\mathcal{W}, \leq, \Vdash, \mathcal{D}, V)$, $A \in \mathtt{pfml}\, m$, $c \in \mathtt{nat}$, if c does not occur in A and $c \neq 0$, then the following holds for all $w \in \mathcal{W}$ and $d \in \mathcal{D}$:*

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

The case $c = 0$ should be excluded because of its special role in (11). Note also that fresh constants we choose are always bigger than 0 as defined in (10). One can use this lemma to prove the soundness.

Theorem 21 (Soundness) *Given a Kripke model $\mathcal{K} = (\mathcal{W}, \leq, \Vdash_{\mathcal{K}}, \mathcal{D}, V)$, we have*

1. $\Gamma \vdash C$ & $w \Vdash_{\mathcal{K}} \Gamma \ \Rightarrow\ w \Vdash_{\mathcal{K}} C[\mathit{nil}]$.
2. $\Gamma \mid A \vdash C$ & $w \Vdash_{\mathcal{K}} \Gamma$ & $w \Vdash_{\mathcal{K}} A[\mathit{nil}]) \ \Rightarrow\ w \Vdash_{\mathcal{K}} C[\mathit{nil}]$.

Both claims can be proved by a simultaneous induction on the derivation. The unique critical case is the rule (14) for the first claim. Given $d \in \mathcal{D}$, we have to show that $w \Vdash_{\mathcal{K}} A[(x, d) :: \mathit{nil}]$ holds. Consider

$$c := \max\{\mathtt{fresh\_out}(\Gamma), \max(\mathtt{OC}(B)), \max(L)\} + 1.$$

Because c does not occur in Γ, we also have $w \Vdash_{\mathcal{K}_{c,d}} \Gamma$, thus the induction hypothesis says that $w \Vdash_{\mathcal{K}_{c,d}} (A[x\backslash \mathtt{Cst}\, c])[\mathit{nil}]$ which is equivalent to $w \Vdash_{\mathcal{K}} A[(x, d) :: \mathit{nil}]$.

Universal Kripke model: $\mathcal{U} = (\texttt{context}, \subseteq, \Vdash_{\mathcal{U}}, \mathcal{D}, V)$ where

$$\begin{aligned} \mathcal{D} &= \texttt{term} \\ V(c) &= c \\ V(f)(t_1, t_2) &= f\, t_1\, t_2, \end{aligned}$$

where $\Gamma \in \texttt{context}$, $c \in \texttt{nat}$, $f \in \texttt{function}$, $t_1, t_2 \in \texttt{term}$. Moreover,

$$\Gamma \Vdash_{\mathcal{U}} P\, t \quad \text{if} \quad \Gamma \vdash P\, t$$

where $P \in \texttt{predicate}$ and $t \in \texttt{term}$.

Fig. 6 Universal Kripke model

6.3 Strong Completeness and Normalisation by Evaluation

The universal Kripke model $\mathcal{U}$ consists of contexts as worlds, the sub-context relation $\subseteq$, and the provability for atomic sentences, and the constant domain function with the image $\texttt{term}$, the set of well-formed terms of the given language. Cf. Figure 6.

Remark 22 *If we have introduced free variables, then the domain of the universal Kripke model would be the set of closed well-formed terms, i.e., well-formed terms without free variables.*

The interpretation of a pseudo-term or a -formula using terms corresponds exactly to the term-substitution:

Lemma 23 *Let $t \in \texttt{pterm}\, m$ and $\eta \in \texttt{list}\,(\texttt{nat} * \texttt{term})$. The following holds with respect to the universal model $\mathcal{U}$.*

$$t[\eta] = t[.\backslash \eta]_{nil}$$

The proof of the strong completeness can be shown by a structural induction thanks to the destined substitution.

Theorem 24 (Strong Completeness) *Let $A \in \texttt{pfml}\, m$, $\Gamma \in \texttt{context}$ and $\eta \in \texttt{list}\,(\texttt{nat} * \texttt{term})$.*

1. *If $\Gamma \Vdash_{\mathcal{U}} A[\eta]$, then $\Gamma \vdash A[.\backslash \eta]_{nil}$.*
2. *If*

$$\forall (C : \texttt{formula})\, (\Gamma : \texttt{context})\, (\Gamma \subseteq \Gamma' \;\&\; \Gamma' \mid A[.\backslash \eta]_{nil} \vdash C \;\Rightarrow\; \Gamma' \vdash C)$$

holds , then $\Gamma \Vdash_{\mathcal{U}} A[\eta]$.

Using the second claim one can easily show that $\Gamma \Vdash_{\mathcal{U}} \Gamma$. The following completeness theorem would hold only for sentences, i.e., without free variables, when we have considered free variables. This is one of the initial ideas of merging free variables with constants.

Theorem 25 (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 $\Gamma \vdash A$.*

Normalisation by Evaluation A combination of completeness and soundness leads to cut-admissibility.

Theorem 26 (Cut admissibility) *Let A, B be formulae and Γ a context. Then the cut-rule is admissible in* LJT*:*

$$\frac{\Gamma \mid A \vdash B \quad \Gamma \vdash A}{\Gamma \vdash B}\ (Cut) \tag{15}$$

Assume $\Gamma \mid A \vdash B$ and $\Gamma \vdash A$. Then (Soundness) implies that $\Gamma \Vdash_{\mathcal{U}} A$, thus $\Gamma \Vdash_{\mathcal{U}} B$. Now one gets a proof of $\Gamma \vdash B$ in LJT by the Strong Completeness.

Because (Cut) is a semantically sound rule, a composition of (Soundness) and (Strong completeness) normalise any proof with (Cut) to a cut-free proof. Moreover, a program extraction from the composition provides a function program which produces a cut-free proof from a deduction with (Cut).

7 Conclusion

We have investigated a novel style for formalising logical theories which is based on McKinna-Pollack's locally named approach. There are new features such as destined substitution. The adequacy of our formalisation is guaranteed by the fact that the whole formalisation is based on a paper which contains almost the same contents and structure.

The formalisations follows the informal proofs given in Herbelin and Lee [11], no more and no less. The most important feature of our approach is the transparency - the formalisation is the same with a traditional informal way of logical arguments. In addition, we exploited the fact that free variables and constants behave technically the same in the proof to merge them.

The question of adequacy of our representation can be answered along the description given in Harper et al. [8] and in Aydemir et al. [1]: Our formal presentation expresses the standard understanding of the rigorous informal system.

– The whole process of our formalisation in Coq fairly corresponds to that of the informal proof of consistency and completeness of LJT with respect to Kripke semantics. Cf. Herbelin and Lee [11].
– Our formal presentation has almost the same structure of formalisations of the same results compared to other approaches we tried also. This also indicates that we do not need to show extra that our formal presentations are equivalent to the informal, conventional one on paper.
– Our formalisation is carried out only by using basic techniques of Coq, no complicated concepts and no weird tactics. We have just followed our intuition as in doing the usual mathematical logic.

The Coq codes are available:

– `http://ropas.snu.ac.kr/~gslee/NbE-merging-free.tar.gz` for the formalisation with the merger of free variables and constants;
– `http://ropas.snu.ac.kr/~gslee/NbE-with-free.tar.gz` for that with both free variables and constants.

References

1. Aydemir, B.E., Charguéraud, A., Pierce, B.C., Pollack, R., Weirich, S.: Engineering formal metatheory. In: POPL, pp. 3–15. ACM Press (2008)
2. Barendregt, H.P.: The Lambda Calculus, Its Syntax and Semantics, *Studies in Logic and the Foundations of Mathematics*, vol. 103. North-Holland (1984)
3. 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)
4. 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)
5. 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)
6. Geuvers, H.: Logics and Type Systems. Ph.D. thesis, University of Nijmegen (1993)
7. Gordon, A.D.: A mechanisation of name-carrying syntax up to alpha-conversion. In: Joyce, J.J., Seger, C.J.H. (eds.) HUG '93, *Lecture Notes in Computer Science*, vol. 780, pp. 413–425. Springer (1994)
8. Harper, R., Honsell, F., Plotkin, G.: A framework for defining logics. J. Assoc. Comput. Mach. **40**(1), 143–184 (1993)
9. 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)
10. 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)
11. 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)
12. Kripke, S.: A Completeness Theorem in Modal Logic. J. Symb. Log. **24**(1), 1–14 (1959)
13. Kripke, S.: Semantical considerations on modal and intuitionistic logic. Acta Philos. Fennica **16**, 83–94 (1963)
14. Krivine, J.L.: Lambda-calcul. Études et Recherches en Informatique. [Studies and Research in Computer Science]. Masson, Paris (1990). Types et modèles. [Types and models]
15. McBride, C.: Dependently Typed Functional Programs and their Proofs. Ph.D. thesis, University of Edinburgh (1999)
16. McKinna, J., Pollack, R.: Pure type systems formalized. In: TLCA, pp. 289–305 (1993)
17. McKinna, J., Pollack, R.: Some lambda calculus and type theory formalized. Journal of Automated Reasoning **23**(3-4), 373–409 (1999)
18. O'Connor, R.: Essential Incompleteness of Arithmetic Verified by Coq. In: TPHOLs, pp. 245–260 (2005)
19. Pitts, A.M.: Nominal logic, a first order theory of names and binding. Inf. Comput. **186**(2), 165–193 (2003)
20. Sato, M., Pollack, R.: External and internal syntax of the lambda-calculus. To appear in Journal of Symbolic Computation
21. Stoughton, A.: Substitution revisited. Theor. Comput. Sci. **59**, 317–325 (1988)
22. Troelstra, A.S., van Dalen, D.: Constructivism in Mathematics Volume I, *Studies in Logic and the Foundations of Mathematics*, vol. 121. North-Holland (1988)