Under consideration for publication in J. Functional Programming

When locally-named becomes nominal

GYESIK LEE
Hankyong National University, Anseong, South Korea
gslee@hknu.ac.kr

HUGO HERBELIN
INRIA, Paris, France
Hugo.Herbelin@inria.fr

SUNYOUNG KIM
Yonsei University, Seoul, South Korea
sunyoungkim831@gmail.com

Abstract

This paper introduces a representation style of variable binding using dependent types. Our representation is a variation of the Coquand-McKinna-Pollack's locally-named representation. The main characteristic of our representation is the use of dependent families in defining expressions such as terms and formulas. We also showed some hidden utility of simultaneous substitution and simultaneous renaming. Consequently, we can deal with definitions and proofs in a more compact way, among which well-formedness, provability, soundness, and completeness. In order to confirm the feasibility of our idea we made several experiments using the proof assistant Coq which supports all the functionalities we need: intentional type theory, dependent types, inductive families, and simultaneous substitution. Moreover, we could reveal a new aspect of locally-named representation style that it can be regarded as a nominal approach under the condition that proofs-as-programs is not a part of the domain of the discourse.

1 Introduction

Around the turn of the 20th century, mathematicians and logicians were interested in the exacter investigation of the foundation of mathematics and soon realized that ordinary mathematical arguments can be represented in formal axiomatic systems. One of the prominent figures in this research was Gottlob Frege. His main concern was twofold: Firstly, whether arithmetical judgments can be proved in a purely logical manner. Secondly, how far one could go in arithmetic by merely using the laws of logic. In *Begriffsschrift* (Frege, 1879), he invented a special kind of language system where statements can be proved as true based only upon some general logical laws and definitions. Then in the two volumes of *Grundgesetze der Arithmetik* (Frege, 1893, 1903) he applied it to provide a formal system where second order arithmetic can be developed.

Although this system is known to be inconsistent, it contains all the essential materials necessary to provide a fundamental basis for dealing with propositions of arithmetic based on an axiomatic system. Indeed it addresses all the *three types of concern that can attend*

a mathematical proof which are mentioned in (Avigad & Harrison, 2014): whether the methods it uses are appropriate to mathematics, whether the proof itself represents a correct use of those methods, and whether a proof delivers an appropriate understanding of the mathematics in question.

A main characteristic of Frege's approach is that truth of statements can be systematically proved using logical laws and axioms. That is, a rigorous and detailed proof can be given for each true statements such that every logical inference can be checked when necessary. This is the main factor why people say that Frege's work initiated an era of applying rigorous scientific method for mathematics. Indeed, since his work, logicians and mathematicians started to consider mathematical systems as axiomatic ones, among which Peano's *The principles of arithmetic* (Peano, 1889), Hilbert's *Grundlagen der Geometrie* (Hilbert, 1899), Russel and Whitehead's *Principia Mathematica* (Whitehead & Russell, 1910, 1912, 1913), Zermelo's axiomatic set theory (Zermelo, 1908), and Church's type theory (Church, 1940). We refer to van Heijennoort's *From Frege to Gödel* (van Heijenoort, 1967) for more about Frege's influence on the development of modern logic and mathematics.

In this article, we focus on an aspect of the tradition of applying a rigorous scientific method for mathematics, namely, formal proof. A formal proof is a proof which is written in an artificial language and in which every step of the inference can be checked according to some fixed logical rules and axioms. Note that this is exactly what Frege had in mind when he developed his system. Indeed, Frege said as follows:

> The gaplessness of the chains of inferences contrives to bring to light each axiom, each presupposition, hypothesis, or whatever one may want to call that on which a proof rests; and thus we gain a basis for an assessment of the epistemological nature of the proven law. (Frege, 2013, p. VII).

The only difference between his idea and the present-day practice is that machine has become ripe enough to assist human in writing down and proving mathematical statements. There are various computer programs that can check and (partially) construct proofs written in their specific programming languages.

More concretely, this paper addresses several issues about the *use* and *role* of variables in proving meta-theories of first-order predicate logic. And some of our ideas go back to Frege's understanding of variables in *Begriffsschrift* (Frege, 1879).

1.1 Motivations

Frege's notion of variables Frege distinguished between two kinds of *signs* when he explained the basic building blocks for constructing syntactic entities like propositions and proofs:

> I therefore divide all signs that I use into *those by which we may understand different* and *those that have a completely determinate meaning*. The former are *letters* and they will serve chiefly to express *generality*. But, no matter how indeterminate the meaning of a letter, we must insist that throughout a given context the letter *retain* the meaning once given to it. (Frege, 1879, p. 11)

Letters represent some objects like numbers or functions left indeterminate and are nowadays called *variables*. And signs that have a completely determinate meaning correspond to function symbols including constants in modern terminology of logic.[1] He then distinguished further between two sorts of letters. He made use of Latin letters a, b, c, etc to express universal validity of propositions, as in

$$(a+b)c = ac+bc$$

and replaced them by old German letters $\mathfrak{a}$, $\mathfrak{b}$, $\mathfrak{c}$, etc in order to state the generality of judgments, as in

$$\vdash \forall\mathfrak{a}\forall\mathfrak{b}\forall\mathfrak{c}\,[\,(\mathfrak{a}+\mathfrak{b})\,\mathfrak{c} = \mathfrak{a}\;\mathfrak{c}+\mathfrak{b}\;\mathfrak{c}\,].$$

In the modern terminology, Latin letters are called *parameters*[2] or *free* variables while German letters are called *local* variables. This is the reason why people mention Frege as the first who used two disjoint sets of variables, see e.g. (Sato & Pollack, 2010, p. 599).

Distinguishing between two kinds of variables is also applied later by (Gentzen, 1934) and (Prawitz, 1965). Local variables play the role of *delimiting the scope* that the generality indicated by the letters cover. Indeed, their role is to remember the places within their scope where "something else" might be substituted, resulting in a less general judgment. On the other hand, parameters syntactically play no essential role in Frege's work except when they are replaced by local variables in stating the generality of judgments. General substitution, for instance, is only performed when instantiating local variables, that is, when making statements less general.

This is the point where we believed that there are some aspects of the role of parameters which might be interesting in relation to mechanization of meta-theories of predicate logic.

Mechanization of logical meta-theories In predicate logic, two sorts of variable binding are involved: Binding local variables is used for representing universal quantification, as in

$$\vdash \forall x P(x)\,,$$

and binding parameters is used for representing parametric derivations, as in

$$A(a) \vdash B(a)\,.$$

In traditional mathematical usage, it is very common to use the same set of variables for both bindings. However, this common practice is not so practical for the use in a mechanical development of a formal metatheory. The main issue with this approach is that local variables can sometimes clash with parameters. For instance, the standard definition of unrestricted substitution for the lambda calculus by (Curry & Feys, 1958, p. 94) causes that variable capture can occur during substitution and that many proofs involving substitution are notoriously tedious because substitution is not defined by a structural induction. A typical way of addressing this issue is to work with α-conversion.

[1] Frege himself rejected to use of the word "variable" since it was hardly possible for him to define it properly. See the footnote by Jourdain on page 10 in (Frege, 1879).

[2] Parameters are sometimes called *global* variables.

However, dealing with α-conversion formally has turned out to be not so feasible because it requires huge amount of extra work. One exceptional work is the nominal techniques in Isabelle/HOL (Urban, 2008) which is based on the nominal logic by Pitts et al. in (Gabbay & Pitts, 2002; Pitts, 2003). However, to the best of our knowledge, there is no user-friendly work dealing with α-conversion formalized in an intentional proof assistant such as Coq, Agda, etc.

People, however, recognized that Frege's idea of distinguishing between the two sorts of variables can be practically applied in doing machine-checked proofs. (Coquand, 1991) suggests to use the same idea as Frege's in order to avoid the need to reason about α-conversion. Following his suggestion, (McKinna & Pollack, 1993, 1999) extensively investigates the main characteristics of using two sorts of variables in proving the meta-theories of lambda calculus and Pure Type Systems. One of the achievements of their great effort is that many important properties of typed lambda calculus can be stated and proved without referring to α-conversion, among which Church-Rosser, standardization and subject reduction.

Another well-known encoding technique of distinguishing between local variables and parameters is the *locally nameless* representation. It also uses two sorts of variables, but it uses names only for parameters while de Bruijn indices are used for local variables. The locally nameless technique was first experimented by (Leroy, 2007). A larger-scale case studies using this technique were conducted by (Aydemir *et al.*, 2008). The usefulness of the locally nameless technique is well summarized by (Charguéraud, 2012). Further information about the importance of encoding techniques especially when variable binding is involved can be found in the POPLmark Challenge project (Aydemir *et al.*, 2005).

Constants as parameters We mentioned above that parameters syntactically play no essential role in Frege's work except when they are replaced by local variables in stating the generality of judgments. This fact can be seen in the following two rules about $\forall$-quantification where variable binding occurs:

$$\frac{\Gamma, A(t) \vdash C}{\Gamma, \forall x A(x) \vdash C}\ (\forall_L) \qquad \text{and} \qquad \frac{\Gamma \vdash A(a) \qquad a \text{ fresh in } \Gamma, A}{\Gamma \vdash \forall x A(x)}\ (\forall_R)\,.$$

They are two rules in the Gentzen-style sequent calculus representing respectively the left and right introduction rules of $\forall$-quantification. And they are the only rules where instantiation of a bound local variable occurs. Note that instantiation of the $\forall$-quantifier by an arbitrary term occurs only in $(\forall_L)$ while in $(\forall_R)$ a local variable is instantiated by a parameter. As we will see later in Figure 3 of the whole presentation of the cut-free LJT, an intuitionistic Gentzen-style sequent calculus, it is unnecessary to consider instantiation of parameters in the definition of the deduction system.

One may 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, 1994, 2002) 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. This fact can also be observed in (McKinna & Pollack, 1999; Leroy, 2007), namely in the proof of the

subject reduction property which requires the substitution lemma of the following form:

$$\frac{\Gamma \vdash N : A \quad \Gamma, a : A, \Delta \vdash M : B}{\Gamma, ([N/a]\Delta) \vdash [N/a]M : [N/a]B}$$

Here one can see that parameters used in the construction of proof terms play the role of binding in the sense that the variable a occurring in M belongs to the scope of a from the environment $\Gamma, a : A, \Delta$. This is the reason why parameters are sometimes called *global* variables. In both work, the substitution lemma is given a name. In (McKinna & Pollack, 1999), it is called `substitution_lem`. In case of (Leroy, 2007), the substitution lemma divided into two versions. One for the terms, the other for types. The lemmata `has_type_stable_type_subst` and `has_type_stable_term_subst` state the corresponding versions.

On the other hand, when the proofs-as-programs correspondence is not on the program, the right introduction rule of implication can hardly be seen as a form of binding. That is, if derivability is a predicate and not a part of the domain of the discourse, there is no need to consider instantiation of parameters. Moreover, it is a commonly accepted fact in proof theory that opening a binder to replace its bound variable with a fresh parameter amounts to expanding the language with a new constant. This point might become more clear when one think of the correspondence between syntax and semantics of the first-order predicate logic. Given a model of a theory, the interpretation of a constant is a fixed object of the domain of the given model. But it depends on the theory. Therefore, the interpretation of a fresh constant can be made arbitrary without affecting the meaning of the given theory. We also make use this fact later, see Lemma 3.3.

This observation gives rise to a question whether we need parameters at all when we formalize meta-theories of a logic system. And it drove us to check whether we could show the same meta-theoretic results even when we do not involve parameters to the language.

1.2 Contributions of the paper

The first contribution of this paper is an answer to the question as explained in the motivation part. That is, we have confirmed the feasibility of using no parameters and letting constants play the role of parameters in formalizing logical metatheory when proofs-as-programs correspondence is not on the program. As our target, we took the formalization of a Kripke-based semantical cut-elimination with respect to LJT.

From the logical point of view, what we have done is very similar to what (Coquand, 1994, 2002) did where she used semantic normalization proof for the implicational propositional calculus. A main difference is that the derivability is a part of the domain of the discourse in her work where β-like reduction on the proofs is an important factor. On the other hand, it is different in our case even though we work with predicate logic with quantification where variable binding occurs. A version of the substitution lemma is still necessary in order to prove the completeness of LJT. However, it does not involve any instantiation of parameters.

The second contribution of this paper is that we provide a reasonable application of dependent type programming in representing language syntax of a predicate logic. The core of our idea lies in the definition of terms and formulas. They are defined by dependent

families. Let m be a list of variables. Then `term` m (resp. `formula` m) denotes a family of terms (resp. formulas) where variables from m possibly occur unbound. That is, the list m, which we call *trace*, collects the local variables that possibly occur *unbound* in a term or a formula although they are supposed to be bound by a $\forall$-quantifier.[3] Then the family `term` *nil* (resp. `formula` *nil*) denotes the set of all well-formed terms (resp. formulas). Here, *nil* is a notation for the empty list. Consequently, one can give a more natural representation of e.g. the derivability predicate without referring to well-formedness extra. This is a major difference from the style of dealing with well-formedness in (McKinna & Pollack, 1993, 1999). We give a full explanation of how we applied this idea to the formalization of our target. We also check out advantages and disadvantages of using dependent families.

The third contribution of this paper consists in strengthening the importance of simultaneous substitution. Some people have already showed the importance of simultaneous substitution in formal reasoning about languages with binding, among which (Stoughton, 1988), (Coquand, 2002), and (McKinna & Pollack, 1999; Pollack, 2006; Pollack *et al.*, 2012). Firstly, we show that for our work simultaneous substitution is not a choice, but a must. Moreover, we present some ways how to work with simultaneous renaming when the choice for a quantification style matters, see the remark after Theorem 3.6. Secondly, we discuss how simultaneous substitution can be used in saving infrastructure for a formal reasoning, even with the locally nameless techniques where numbers are used for local variables instead of names, see Section 5.

The last contribution of our work is related to the nominal representation. After having finished our work, we noticed that we can give a different interpretation of `term`m and `formula`m from that given above in the part of the second contribution without changing anything in the original formalization. And the new interpretation results in a nominal representation. In summary, we show that when the Coquand-McKinna-Pollack style locally-named representation is used without parameters, it results in a nominal representation. The new interpretation is explained in Section 4 after all the other contributions have been explained.

1.3 About mechanization

We used the proof assistant Coq (Coq Development Team, 2015) as the programming tool. Coq provides all the functionalities we need in order to realize our ideas: intentional type theory, dependent types, inductive families, and simultaneous substitution.[4]

This paper mainly explains the version with traces in the definition of dependent families of terms `term` m and formulas `formula` m where no parameters are allowed. The formalization of the version both with traces and parameters almost identical.

We did also several experiments with other representation styles in order to check the utility and feasibility of our ideas, among which simultaneous substitution, simultaneous

[3] We remark that our idea follows the usage, common in the theory of lambda calculus, to have a notation for the set of terms over some set of variables.

[4] The Coq proof scripts are available online. Please visit `http://formal.hknu.ac.kr/jfp/`.

renaming, quantification style, comparison between simultaneous renaming and swapping variables. More detail will be given in Section 5.

1.4 Outline of the paper

Section 2 presents the syntactic part of an intuitionistic predicate calculus LJT and discusses technical details of our formalization, among which simultaneous substitution, renaming, and quantification style. Section 3 introduces a Kripke semantics for LJT and explains the role of simultaneous substitution in establishing meta-theoretic results about LJT such as soundness, completeness, and cut-admissibility. Section 4 explains our contribution how to get a nominal representation as a variation of the Coquand-McKinna-Pollack style locally-named representation. Section 5 gives an overview over the whole work what we did in addition to that we mainly discuss in this paper and discuss advantages and disadvantages of ideas we suggest.

2 Presentation of the intuitionistic sequent calculus LJT

As is well known, the nominal representation of using one sort of variables is not so feasible in doing formal proofs in an intentional theorem prover when variable binding is involved. Still there were some trials to show the feasibility of the nominal representation style such as Stump's partial contribution to the POPLmark Challenge. However, in a bigger and more complicated scale, the notorious problem with variable capture remained unsolved except for the nominal techniques in Isabelle/HOL (Urban, 2008) which is based on the nominal logic by Pitts et al. (Gabbay & Pitts, 2002; Pitts, 2003). On the other hand, when one works with an intentional proof assistant, the locally-named representation (McKinna & Pollack, 1993, 1999) and the locally nameless representation (Aydemir *et al.*, 2008; Charguéraud, 2012) are excellent choices. Each style has it own advantages and disadvantages, but our interest lies in the locally-named representation because of its use of named variables for binding.

Our work started with the observation that both representations are too permissive in definition of terms. Some terms could not have meaning because they contain free occurrences of some local variables which are subject to be bound. This gives rise to the necessity of an extra syntax for the so-called *well-formed* terms. And it is a distinctive feature of McKinna and Pollack's work that they introduced a way of avoiding appeal to such well-formedness considerations except in a very small number of places such as the definition of typing rules. Moreover, their idea is also well adapted to handle the same problem in the locally nameless representation.

Remark 2.1
There are approaches with two sorts of variables that require no extra syntax for well-formedness. (Sato & Pollack, 2010), 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 consider such approaches because the mechanism they use is not related to our concern.

In order to remove the necessity of using extra syntax for well-formed terms at all, we use *traces* to control the information of local variables occurring in the construction of terms. Our idea is very close to that used in the typechecker example by (McBride & McKinna, 2004, Section 7) and the use in Agda of families indexed by Nat to represent well-scoped terms. And the idea appears to go back to (Bird & Paterson, 1999), (Bird & Meertens, 1998), and (Altenkirch & Reus, 1999). In Section 5, the relationship with McBride and McKinna's work will be explained in more detail.

2.1 A predicate language without parameters

As explained in the introductory part, parameters seem to play no essential roles when the proofs-as-programs correspondence is not on the program. We also wanted to check this point and applied our idea to the formalization of a Kripke-based semantical cut-elimination of the intuitionistic first-order predicate logic, called LJT. We use a version of the locally-named representation. A main characteristic of the language of LJT is that it contains local variables, but no parameters.

The language of LJT involves two kinds of expressions, namely terms and formulas. And the definition of formulas involves universal quantification which is a kind of variable binding. Therefore, we believed that it provides an appropriate case study to test and confirm our ideas.

We adopt sequent calculus style derivability to represent proofs. The advantage of such an approach is that it has an easy-to-define notion of the normal form. A proof is in normal form when it is merely constructed without using the cut rule.

The language we consider contains $\rightarrow$ and $\forall$ as the sole connectives. As for the non-logical symbols, we assume that the language contains unary predicates symbols, binary function symbols, and infinitely many constant symbols. Note that this assumption is not a real restriction. Firstly, every language can be conservatively extended to a language with infinitely many constants. Secondly, functions or predicates of other arities can be represented by using binary function symbols.

We use names to represent both local variables and constants. Letters like c, d, c_i, d_i vary over constants while letters like x, y, x_i, y_i vary over variables. In addition, f, g, f_i, g_i (resp. P, Q, P_i, Q_i) denote function (resp. predicate) symbols.

Remark 2.2

All the sets here mentioned are supposed to be *decidable*. A set X is *decidable* if, constructively, $\forall u, v \in X\,(u = v \vee u \neq v)$ holds, i.e., if there exists a decision procedure to distinguish between $u = v$ and $u \neq v$ for any two elements of X.

For the formalization, we use (finite) lists to denote finite sets of constants, variables, or formulas. $\{x_1, ..., x_n\}$ stands for the list $x_1 :: \cdots :: x_n :: nil$. For our purpose, it is sufficient to define a *sublist* relation in a set-theoretic manner: A list ℓ is a sublist of another list k if ℓ is a subset of k when they are regarded as finite sets. We also use the usual set-theoretic notations such as $\in$, $\notin$, $\subseteq$, etc.

Terms:

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

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

Formulas:

$$\frac{P \in \texttt{predicate} \quad t \in \texttt{term}\, m}{\texttt{Atom}\,(P,t) \in \texttt{formula}\, m} \qquad \frac{A \in \texttt{formula}\, m \quad B \in \texttt{formula}\, m}{A \to B \in \texttt{formula}\, m}$$

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

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

Occurrence of variables:

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

Occurrence of constants:

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

Fig. 1. Terms and formulas without parameters

2.2 Dependent families of terms and formulas

As mentioned before, one of our main ideas is to define terms and formulas as dependent families.

The type `name` denotes the set of names. Given a list m of names, the type $\texttt{term}\, m$ (resp. $\texttt{formula}\, m$) denotes the set of terms (resp. formulas), where local variables from m possibly occur unbound although they are supposed to be bound by a $\forall$-quantifier, see Figure 1.

The side condition $(h : x \in m)$ in the definition of $\texttt{Var}\, x\, h \in \texttt{term}\, m$ is crucial. In this manner, we control the information on variables used in the construction of terms and formulas. Indeed, every variable occurring in a term or a formula occurs already in the trace:

Lemma 2.3
Let $e \in \texttt{term}\, m$ or $e \in \texttt{formula}\, m$. Then, $\texttt{OV}(e) \subseteq m$.

Consequently, the set of well-formed terms (resp. formulas) can be syntactically represented by $\texttt{term}\, \mathit{nil}$ (resp. $\texttt{formula}\, \mathit{nil}$).

2.3 Substitution and trace relocation

The definition of the substitution is a part to which we payed special attention. There are two reasons. Firstly, in order to establish in a natural way the soundness and the

Let $\eta = (x_1,u_1),...,(x_n,u_n)$ be an association, where $u_i \in \mathtt{term}$. Suppose further $t \in \mathtt{term}\,m$ and $A \in \mathtt{formula}\,m$.

1. $[\ell \Uparrow \eta]t \in \mathtt{term}\,\ell$ is recursively defined:

$$\begin{array}{rcl} [\ell \Uparrow \eta](\mathtt{Var}\,y\,h) & = & \begin{cases} \mathtt{Var}\,y\,h' & \textit{if } y \in \ell \\ \mathtt{treloc}\,u_j\,h_j & \textit{if } y \notin \ell \textit{ and } j = \min\{i : y = x_i\} \\ \mathtt{Cst}\,0 & \textit{otherwise} \end{cases} \\ [\ell \Uparrow \eta](\mathtt{Cst}\,c) & = & \mathtt{Cst}\,c \\ [\ell \Uparrow \eta](\mathtt{App}\,f\,t_1\,t_2) & = & \mathtt{App}\,f\,([\ell \Uparrow \eta]t_1)\,([\ell \Uparrow \eta]t_2) \end{array} \tag{1}$$

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

2. $[\ell \Uparrow \eta]A \in \mathtt{formula}\,\ell$ is recursively defined:

$$\begin{array}{rcl} [\ell \Uparrow \eta](Pt) & = & P([\ell \Uparrow \eta]t) \\ [\ell \Uparrow \eta](A \to B) & = & [\ell \Uparrow \eta]A \to [\ell \Uparrow \eta]B \\ [\ell \Uparrow \eta](\forall x B) & = & \forall x([x :: \ell \Uparrow \eta]B) \end{array} \tag{2}$$

Fig. 2. Simultaneous substitution for terms and formulas

completeness of LJT with respect to a Kripke semantics, it is necessary to work with a simultaneous substitution, see Theorem 3.1 and Theorem 3.6.

Secondly, because of the trace part, it is not clear to which family the result of a substitution should belong. Suppose $t \in \mathtt{term}\,m$ and $s \in \mathtt{term}\,m'$. There are infinitely many families to which the result of the substitution of s for a variable in t could belong. Any term family $\mathtt{term}\,\ell$ such that $\mathtt{OV}(t), \mathtt{OV}(s) \subseteq \ell$ can be chosen. We then decided to define substitution such that it respects the following two points:

- Variables supposed to be subsequently bound in a formula should not be considered generally substitutable for.
- Only well-formed terms have a real meaning.

The idea is as follows. We first declare a trace ℓ of variables prohibited from being substituted and then substitute only well-formed terms. The role of ℓ is well demonstrated in the abstraction case (2) where the trace is extended by a bound variable x in order to forbid any substitution for x.

The point is that we know before the substitution is performed where the resulting term will arrive at. In particular, if $\ell = nil$ then the result of a substitution is a well-formed term or a well-formed formula. Later we will see that this makes us work with more intuitive definitions and proofs, among which the inference rules in Figure 3 and the Universal Completeness in Theorem 3.6.

For the definition of simultaneous substitution, we use lists of pairs of variables and well-formed terms. Such lists are called *associations*. Associations will also be used later in the semantic part.

Suppose e is a term or a formula. Let ℓ be a trace and $\eta = (x_1,u_1),...,(x_n,u_n)$ an association with $u_i \in \mathtt{term}\mathit{nil}$ for all i. Then

$$[\ell \Uparrow \eta]\, e$$

denotes the result of simultaneously substituting u_i for x_i in e. The simultaneous substitution operation is defined by a structural recursion as in Figure 2.

Notations. We handle the single substitution $[\ell \Uparrow u/x]\, e := [\ell \Uparrow (x,u)]\, e$ as a special case. Furthermore, we write $[u/x]\, e$ when $\ell = \mathit{nil}$ for better readability.

Two points should be mentioned about the definition. Firstly, some variables are ignored by assigning them $\mathtt{Cst}\,0$ as in (1). All the free occurrences of variables are supposed to be covered either by the list ℓ or by the domain of an association. However, we decided to ignore some variables in order to argue in a more general setting.

The ignored case could be handled differently, namely by including appropriate extra propositional arguments witnessing the side condition that $\mathtt{OV}(e) \subseteq \mathsf{dom}(\eta)$ or $\mathtt{OV}(A) \subseteq \mathsf{dom}(\eta)$ holds. This would make our work more perfect for an application of dependently typed programming. However, there are several reasons for our choice. More detail will be given later in Section 5 where our mechanization experience is explained. Here we just say, the definition given above works most smoothly from the technical point of view.

Secondly, we had to use trace relocation in (1). The substituted term u_j is of type $\mathtt{term}\mathit{nil}$. In order to make typechecking work, we need to relocate it to $\mathtt{term}\,\ell$, and this is the only reason why we need trace relocation.

In the following, notations are simplified for better readability. Given two traces m and ℓ, the trace relocation operation $\mathtt{treloc} : \mathtt{term}\,m \to \mathtt{term}\,\ell$ is a partial function defined only for terms t such that $\mathtt{OV}(t) \subseteq \ell$:

$$\begin{aligned} \mathtt{treloc}(\mathtt{Var}\,x\,h) &= \mathtt{Var}\,x\,h' \\ \mathtt{treloc}(\mathtt{Cst}\,c) &= \mathtt{Cst}\,c \\ \mathtt{treloc}(\mathtt{App}\,f\,t_1\,t_2) &= \mathtt{App}\,f\,(\mathtt{treloc}(t_1))\,(\mathtt{treloc}(t_2)) \end{aligned} \tag{3}$$

where h' is a proof of $x \in \ell$, which can be obtained from $\mathtt{OV}(t) \subseteq \ell$.[5]

The relocation function is homomorphic in the sense that it does not change or bother any functionality of terms, both syntactically and semantically. Note just that the proof element in the definition of a term is inessential so long as the trace contains all the necessary variables and that $\mathtt{treloc}(t)$ does not change anything but the proof part element. In order to demonstrate that the substitution behaves as expected, we mention two lemmata.

Lemma 2.4 (*Substitution Lemma*)
Given a trace ℓ, let e be a term or a formula, $u \in \mathtt{term}$, and η an association. Then

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

Lemma 2.5 (*Relocation has no impact on substitution*)

[5] The choice of h' in (3) is irrelevant. One could assume the proof irrelevance axiom or can show the uniqueness of the membership relation. It works well with Coq.

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

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

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

Fig. 3. Cut-free LJT

Given a trace ℓ, let t be a term such that $\mathtt{OV}(t) \subseteq k$, and η an association. Then

$$[\ell \Uparrow \eta](\mathtt{treloc}(t)) \quad = \quad [\ell \Uparrow \eta]t\,.$$

2.4 Cut-free LJT and weakening

The Gentzen-style sequent calculus LJT presented in Figure 3 is obtained from the intuitionistic sequent calculus LJ by restricting the use of the left introduction rules. A sequent is either of the form $\Gamma \mid A \vdash C$ or of the form $\Gamma \vdash C$, where only well-formed formulas are involved. The location between the vertical bar "|" and the sign "$\vdash$" is called *stoup* and contains the principal formula of the corresponding left introduction rule.

Note that the inference rules can be represented exactly as in Figure 3 without including any side condition since a context is of type $\mathtt{list}(\mathtt{formula}\mathit{nil})$ and a well-formed formulas is of type $\mathtt{formula}\mathit{nil}$.

Remark 2.6
(Herbelin, 1994, 1995) and (Mints, 1996) showed that cut-elimination matches normalization in the $\overline{\lambda}$-calculus, which is a variant of λ-calculus for the sequent calculus structure. This implies that LJT supports well the proofs-as-programs correspondence.

The right quantification rule $(\forall_R)$ requires some explanations. Note first that a fresh constant c is used in the premise of the rule. This is the point witnessing the fact that a fresh constant can be used instead of a fresh parameter.

The next one we need to explain is about our choice for the quantification style. It is enough for the premise of $(\forall_R)$ to hold for *one* fresh constant. There are some issues about this style of quantification such as the fact that it provides too weak an induction principle. For example, let us try to prove weakening in the following form:

> Suppose $\Gamma \subseteq \Gamma'$ and $\Gamma \vdash A$. Then $\Gamma' \vdash A$.

If one tries to prove this lemma by induction on the given deduction, one soon notices that a renaming lemma of the following form is necessary:

> If $\Delta \vdash [c/x]A$ holds for a fresh constant c, then $\Delta \vdash [d/x]A$ holds for every fresh constant d.

However, another naive trial to prove it would lead to a vicious circle that now weakening is necessary. An excellent solution to break this vicious circle is provided in (Pitts,

2003). He showed that using swapping one can easily prove renaming without appealing to weakening.

Here we explain another option which enables us to prove weakening and renaming all together. Proving weakening and renaming simultaneously seems to be a natural idea since they are somehow mutually dependent. Our idea is to use simultaneous renaming which is a generalized form of variable swapping.

Simultaneous renaming is a kind of simultaneous substitution where in our case constants are replaced with constants.[6] In the following, $\rho = (c_1, d_1), ..., (c_n, d_n)$ stands for a simultaneous renaming. Given a formula $A \in \texttt{formula}m$, ρA denotes the formula of type $\texttt{formula}m$ where each constant c_i occurring in A is simultaneously renamed to d_i. Then $\rho\,\Gamma$ is canonically defined for a context Γ. Now we can show the following generalized version of weakening which can be proved by a simple, structural induction. Weakening and renaming are special forms of this version.

Theorem 2.7 (*Generalized Weakening*)
Let A, C be well-formed formulas, Γ, Γ' contexts such that $\Gamma \subseteq \Gamma'$, and ρ an arbitrary renaming. Then the following holds:

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

Note that no side conditions are imposed on the renaming ρ. Even injectivity is not required while it is the case with swapping which handles special permutation of variables. Again this is because derivability is predicate and does not belong to the part of the domain of the discourse. Otherwise, some kind of bijectivity of the renaming will be necessary as already demonstrated in (McKinna & Pollack, 1999). We also checked where some conditions required for the renaming if the proofs-as-terms correspondence is involved. More detail will be given later in Section 5.

3 Kripke semantics, soundness, completeness, and cut-admissibility

Having seen the basic syntax of LJT and some technically important points for the formalization, we provide in this section a Kripke semantics for LJT.

Kripke semantics was created in the late 1950s and early 1960s by Saul Kripke in (Kripke, 1959, 1963). It was first introduced for modal logic, and later adapted to intuitionistic logic and other non-classical or classical systems (cf. (Troelstra & van Dalen, 1988) and (Ilik *et al.*, 2010)). Here, we use the conventional Kripke model adopted by Troelstra and van Dalen.

A Kripke model $\mathscr{K} = (\mathscr{W}, \leq, \Vdash, \mathscr{D}, V)$ is a tuple of a partially-ordered set $\mathscr{W}$ of *worlds*, a domain $\mathscr{D}$, interpretations of constant and function symbols into the domain, and a relation between worlds, predicates, and domain elements (cf. Figure 4). Interpretation of terms is based on an association $\eta \in \texttt{list}\,(\texttt{name} * \mathscr{D})$. Note that some variables are ignored. This is necessary to cope with the definition of simultaneous substitution where also some variables are ignored. Furthermore, the proof term for a list membership is simply neglected. Consequently, the trace relocation has no impact on the Kripke semantics.

[6] If parameters play their own role, then we have to rename parameters.

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

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

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

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

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

Interpretation of terms: Let $\eta \in \mathtt{list}\,(\mathtt{name} * \mathscr{D})$

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

Here $\eta(x) = d$ if (x,d) is the first occurrence in η from left of the form $(x,\_)$.

Forcing: The relation $\Vdash$ is inductively extended to general formulas.

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

$$w \Vdash \Gamma \quad \textit{iff} \quad w \Vdash A[\mathit{nil}] \textit{ for all } A \in \Gamma$$

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

Fig. 4. Kripke semantics

Soundness and completeness can be formalized without any difficulty.

Theorem 3.1 (*Soundness*)

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

If we had included parameters and let them play their intended role, the soundness proof would be very simple to prove as shown in (Herbelin & Lee, 2009). However, because constants took the role of parameters, the $(\forall_R)$ rule requires more attention.

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

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

holds. At this point, the premise of $(\forall_R)$ 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.

A solution lies in the fact that fresh constants are as good as fresh parameters. Syntactically, this fact is represented by the renaming lemma. At the semantic level, this

corresponds to creating a new Kripke model from a given one such that the semantics remains nearly identical.

Definition 3.2
Given a Kripke model $\mathscr{K} = (\mathscr{W}, \leq, \Vdash, \mathscr{D}, V)$, a constant c, and a value $d \in \mathscr{D}$, we define a new Kripke model $\mathscr{K}_{c,d} := (\mathscr{W}, \leq, \Vdash, \mathscr{D}, V_{c,d})$, where

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

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

Lemma 3.3 (*Forcing with fresh constants*)
Given a formula A and a constant c, if c does not occur in A, then the following holds: For any Kripke model $\mathscr{K} = (\mathscr{W}, \leq, \Vdash, \mathscr{D}, V)$, any $w \in \mathscr{W}$, and any $d \in \mathscr{D}$ we have

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

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 well-formed.)

Now, we use Lemma 3.3 to show that $w \Vdash_{\mathscr{K}_{c,d}} \Gamma$. Consequently, by induction hypothesis, we also have $w \Vdash_{\mathscr{K}_{c,d}} ([c/x]A)[nil]$. Finally, we can prove (6):

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

where the equivalence in (7) follows from the following lemma.

Lemma 3.4
Let A be a formula, u a well-formed term, and ℓ a trace. Then for any Kripke model $\mathscr{K} = (\mathscr{W}, \leq, \Vdash, \mathscr{D}, V)$, any $w \in \mathscr{W}$, and any association η, we have

$$w \Vdash_{\mathscr{K}} ([\ell \setminus \{x\} \Uparrow (x,u) :: nil\,]A)[\eta] \iff w \Vdash_{\mathscr{K}} A[(x, u[\eta]) :: \eta\,],$$

where $\ell \setminus \{x\}$ denotes the trace obtained from ℓ by removing x.

Formalization of completeness part is done in the same way as in (Herbelin & Lee, 2009). That is, we use the fact that LJT is complete with respect to a universal Kripke model $\mathscr{U}$ defined as follows:

Definition 3.5 (*Universal Kripke Model*)
$\mathscr{U} = (\mathtt{context}, \subseteq, \Vdash_{\mathscr{U}}, \mathtt{term}\, nil, V_{\mathscr{U}})$, where

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

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

Note that in the universal model $\mathscr{U}$, the interpretation of terms corresponds to substitution: Given a term $t \in \mathtt{term}\, m$ 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, says that we have a similar correspondence between forcing and deduction.

Theorem 3.6 (*Universal Completeness*)
Let A be a formula, $\Gamma \in \texttt{context}$, and η be an association. Then, $\Gamma \Vdash_{\mathscr{U}} A[\eta]$ implies $\Gamma \vdash [nil \Uparrow \eta]A$.

Note that A used in the universal completeness theorem is an arbitrary raw formula. This fact and the use of simultaneous substitution enable us to prove this natural correspondence between syntax and semantics by a simple structural induction on A.

Now the completeness follows.

Theorem 3.7 (*Completeness*)
Let A be a closed formula and Γ a context. If, for any Kripke model $\mathscr{K} = (\mathscr{W}, \leq, \Vdash, \mathscr{D}, V)$ and any $w \in \mathscr{W}$, $w \Vdash A$ follows from $w \Vdash \Gamma$, then we have $\Gamma \vdash A$.

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

Theorem 3.8 (*Cut-admissibility*)
Let A, B be formulas and Γ a context. Then, (Cut) is admissible in LJT:

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

Proof
Suppose $\Gamma \mid A \vdash B$ and $\Gamma \vdash A$ hold. Then, by Soundness, $\Gamma \Vdash_{\mathscr{U}} A$ and $\Gamma \Vdash_{\mathscr{U}} B$ hold, too. Consequently, $\Gamma \vdash B$ holds by the Universal Completeness. □

Because (Cut) is a semantically sound rule, a composition of (Soundness) and (Universal Completeness) normalizes any proof with (Cut) to a cut-free proof.

4 Locally-named representation as a nominal representation

This section explains our last contribution. We claim that, when the Coquand-McKinna-Pollack style locally-named representation is used without parameters, it results in a nominal representation. In order to support our claim, we only need change the interpretation of the dependent families $\texttt{term}m$ and $\texttt{formula}m$.

Until now, we have used the locally-named representation without parameters. That is, we have used only local variables. Moreover, we have let constants play the role of parameters. The main reason why this worked is that the proofs-as-programs correspondence is not involved.

However, when we look back at the whole formalization, we notice that the local variables used in an expression are divided into two classes: variables that are really bound by a quantifition and variables that are not bound by any quantification and controlled by a trace. Moreover unbound variables behave like "*parameters*" in the conventional style of using one sort of variables although their role is took over by constants.

In summary, it looks like as if we used one sort of variables which are usually called *bound* or *free* depending on their locations in an expression. We just gave no role to "free" variables and let constants play their role instead. This is the reason why we find the following new interpretation plausible:

- $\texttt{term}m$ denotes the family of terms with possible occurrences of *parameters* from the list m.

- `formula`*m* denotes the family of formulas with possible occurrences of *parameters* from the list *m*.
- `term`*nil* denotes the family of *closed terms*, i.e. no occurrences of *parameters*.
- `formula`*nil* denotes the family of *sentences*.

The rest of our work copes well with this new interpretation without changing anything in the formalization. Only the meaning of some notations changes, among which:

- only closed terms can be substituted;
- the inference rules are defined only for sentences;
- the domain of the universal model consists of closed terms.

We emphasize again that this new interpretation works since the role of parameters can be taken over by constants. That is, when one is interested in formalization of a logical metatheory, but not directly in the proofs-as-programs correspondence, then one could work with a nominal representation style as we propose in this paper.

In order to check the plausibility of our proposal, we did another formalization of the same contents. This time we followed the usual style of locally-named representation without using traces. We tried also two versions, one with parameters and the other without parameters. Except some differences made by the absence of traces, the formalization for both versions works almost the same as in the cases with traces. There are nothing special to be mentioned extra except that we confirmed once more that locally-named representation approach suits well in spite of variable binding and that locally-named representation can become nominal in the same way as we demonstrated in this paper.

5 Practical formal metatheory

About using traces. The elements of a trace are not per se relevant, which is reflected by the fact that trace relocation has no impact on substitution and Kripke semantics. The only important thing is their occurrence in the trace, which is tracked by proofs of list membership. This allows names and de Bruijn indices to be superimposed. Indeed their relationship can be observed when we look at the use of de Bruijn indices in McBride and McKinna's typechecker example from (McBride & McKinna, 2004, Section 7):

- The de Bruijn index 0 corresponds to a proof that $x \in x :: m$;
- The de Bruijn successor S on indices corresponds to a proof that $x \in m$ implies $x \in y :: m$.

Another point about using traces is that one can nicely work with syntax, for instance well-formedness and provability. However, it requires a good support of dependently typed programming. In case of Coq, working with dependent types is sometimes heavy-going. And this is one reason why the simultaneous substitution is defined as a kind of partial function. More detail about why it becomes arduous when we define it in a more dependently typed programming style is explained in a technical report which can be found in the homepage of this work.[7] We have not tested yet, but it could work smoothly with other tools such as Agda.

[7] Please visit `http://formal.hknu.ac.kr/jfp/`.

Case studies. In order to check the plausibility of our ideas introduced in this paper, we took several targets to formalize in Coq. Here we give summary of our experience. A detailed explanation with many technical issues is given in the above mentioned technical report.

First, the formalization with traces: In this paper, we introduced only the version without parameters because our main concern was to check the role of parameters when the proofs-as-programs correspondence is not involved. The version with parameters has nothing special to mention. The language there contains parameters, but they play no more role than that of constants as explained in this paper.

Second, Coquand-McKinna-Pollack's locally-named representation style: We took the same target as in the first case, but this time used the Coquand-McKinna-Pollack's locally-named representation style. There are also two versions, one with parameters and the other without parameters. The two versions are all easier to handle because no dependent types involved.

Third, the soundness of the Church-style simply-typed lambda calculus: This is the case where we tested our idea of using simultaneous substitution and simultaneous renaming. We took Leroy's contribution to the POPLmark Challenge (Leroy, 2007) and slimmed it down to handle the simply-typed lambda calculus, still using the locally nameless representation. The main changes we made are as follows.

- Simultaneous substitution instead of single substitution: We checked that simultaneous substitution works well also with de Bruijn indices.
- Simultaneous renaming instead of variable swapping in order to handle weakening and renaming: We wanted to check the utility of simultaneous renaming in dealing with weakening and renaming when the conventional style of quantification is used, that is the quantification style requiring one fresh instantiation. We could show that simultaneous renaming works well when some injectivity condition is imposed. An interesting point is that bijectivity is not required as assumed in (McKinna & Pollack, 1999).

Epilogue and acknowlegements

The main idea of this paper is that the Coquand-McKinna-Pollack style locally-named representation can be used successfully in formalization of logical metatheory with variable binding, especially when the proofs-as-programs correspondence is irrelevant, which is usually the case for logicians and mathematicians. In order to convince the reader, we tried several experiments, and all the experiments we made confirmed that our idea works well in spite of some technical issues.

The point in the last contribution explained in Section 4 is recognized when we pondered on a comment made by an anonymous referee from a previous, unsuccessful trial to a publication. We would like to express our sincerest thanks to him/her for this and many other constructive comments. This paper is revised mainly based on his/her comments.

References

Bibliography

Altenkirch, Thorsten, & Reus, Bernhard. (1999). Monadic presentation of lambda terms using generalized inductive types. *Pages 453–468 of: LNCS 1683*.

Avigad, Jeremy, & Harrison, John. (2014). Formally verified mathematics. *Communications of the acm*, **57**(4), 66–75.

Aydemir, Brian, Charguéraud, Arthur, Pierce, Benjamin C., Pollack, Randy, & Weirich, Stephanie. (2008). Engineering formal metatheory. *Acm sigplan notices*, **43**(January), 3.

Aydemir, Brian E., Bohannon, Aaron, Fairbairn, Matthew, Foster, J Nathan, Pierce, Benjamin C., Sewell, Peter, Vytiniotis, Dimitrios, Washburn, Geoffrey, Weirich, Stephanie, & Zdancewic, Steve. (2005). Mechanized Metatheory for the Masses: The POPLmark Challenge. *Pages 50–65 of: LNCS 3603*.

Bird, Richard, & Meertens, Lambert. (1998). Nested datatypes. *Mathematics of program construction*, **1422**, 52–67.

Bird, Richard S, & Paterson, Ross. (1999). de Bruijn notation as a nested datatype. *Journal of functional programming*, **9**, 77–91.

Charguéraud, Arthur. (2012). The locally nameless representation. *Journal of automated reasoning*, **49**(3), 363–408.

Church, Alonzo. (1940). A Formulation of the Simple Theory of Types. *The journal of symbolic logic*, **5**(2), 56–68.

Coq Development Team. (2015). *The Coq Proof Assistant Reference Manual*. Available at `http://coq.inria.fr`.

Coquand, Catarina. (1994). From Semantics to Rules: A Machine Assisted Analysis. *Lncs 832*, 91–105.

Coquand, Catarina. (2002). A Formalised Proof of the Soundness and Completeness of a Simply Typed Lambda-Calculus with Explicit Substitutions. *Higher-order and symbolic computation*, **15**, 57–90.

Coquand, Thierry. (1991). An algorithm for testing conversion in Type Theory. *Logical frameworks*, 255–279.

Curry, Haskell B., & Feys, Robert. (1958). *Combinatory Logic. Volume 1*. North Holland.

Frege, Gottlob. (1879). Begriffsschrift, a formula language modelled upon that of arithmetic, for pure thought. *Pages 1–82 of:* van Heijenoort, Jean (ed), *From Frege to Godel*.

Frege, Gottlob. (1893). *Grundgesetze der Arithmetik, Volume 1*. Jena: Verlag Hermann Pohle.

Frege, Gottlob. (1903). *Grundgesetze der Arithmetik, Volume 2*. Jena: Verlag Hermann Pohle.

Frege, Gottlob. (2013). *Frege: Basic Laws of Arithmetic*. Oxford University Press.

Gabbay, Murdoch J., & Pitts, Andrew M. (2002). A new approach to abstract syntax with variable binding. *Formal aspects of computing*, **13**(3-5), 341–363.

Gentzen, Gerhard. (1934). Untersuchungen über das logische Schließen. I. *Mathematische zeitschrift*, **39**(2), 176–210.

Herbelin, Hugo. (1994). A Lambda-Calculus Structure Isomorphic to Gentzen-Style Sequent Calculus Structure. *Pages 61–75 of: CSL '94*. Lecture Notes in Computer Science, vol. 933. Springer.

Herbelin, Hugo. 1995 (Jan.). *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.

Herbelin, Hugo, & Lee, Gyesik. (2009). Forcing-Based Cut-Elimination for Gentzen-Style Intuitionistic Sequent Calculus. *Pages 209–217 of:* Ono, Hiroakira, Kanazawa, Makoto, & de Queiroz, Ruy J. G. B. (eds), *WoLLIC*. Lecture Notes in Computer Science, vol. 5514. Springer.

Hilbert, David. (1899). *Grundlagen der Geometrie*.

Ilik, Danko, Lee, Gyesik, & Herbelin, Hugo. (2010). Kripke models for classical logic. *Ann. pure appl. logic*, **161**(11), 1367–1378.

Kripke, Saul. (1959). A Completeness Theorem in Modal Logic. *J. symb. log.*, **24**(1), 1–14.

Kripke, Saul. (1963). Semantical considerations on modal and intuitionistic logic. *Acta philos. fennica*, **16**, 83–94.

Leroy, Xavier. (2007). A locally nameless solution to the POPLmark challenge. *Tecnical report*, **6908**(INRIA).

McBride, Conor, & McKinna, James. (2004). The view from the left. *Journal of functional programming*, **14**(1), 69–111.

McKinna, James, & Pollack, Robert. (1993). Pure type systems formalized. *Pages 289–305 of: TLCA*.

McKinna, James, & Pollack, Robert. (1999). Some lambda calculus and type theory formalized. *Journal of automated reasoning*, **23**(3-4), 373–409.

Mints, Gregory. (1996). Normal forms for sequent derivations. *Pages 469–492 of: Kreiseliana*. Wellesley, MA: A K Peters.

Peano, Jiuseppe. (1889). *The Principles of Arithmetic (Latin)*.

Pitts, Andrew M. (2003). Nominal logic, a first order theory of names and binding. *Information and computation*, **186**(2), 165–193.

Pollack, Randy. (2006). *Talk: Reasoning About Languages with Binding: Can we do it yet?*

Pollack, Randy, Sato, Masahiko, & Ricciotti, Wilmer. (2012). A Canonical Locally Named Representation of Binding. *Journal of automated reasoning*, **49**, 185–207.

Prawitz, Dag. (1965). *Natural Deduction*. Almqvist & Wiksell.

Sato, Masahiko, & Pollack, Randy. (2010). External and internal syntax of the lambda-calculus. *Journal of symbolic computation*, **45**(5), 598–616.

Stoughton, Allen. (1988). Substitution Revisited. *Theoretical computer science*, **59**, 317–325.

Troelstra, Anne S., & van Dalen, Dirk. (1988). *Constructivism in Mathematics: An Introduction I and II*. Studies in Logic and the Foundations of Mathematics, vol. 121, 123. North-Holland.

Urban, Christian. (2008). Nominal Techniques in Isabelle/HOL. *Journal of automated reasoning*, **40**(4), 327–356.

van Heijenoort, Jean. (1967). From Frege to Goedel.

Whitehead, Alfred North, & Russell, Bertrand. (1910). *Principia mathematica, Volume 1*. Cambridge University Press.

Whitehead, Alfred North, & Russell, Bertrand. (1912). *Principia mathematica, Volume 2*. Cambridge University Press.

Whitehead, Alfred North, & Russell, Bertrand. (1913). *Principia mathematica, Volume 3*. Cambridge University Press.

Zermelo, Ernst. (1908). Untersuchungen ueber die Grundlagen der Mengenlehre. I. *Mathematische annalen*, **65**.