J. Korean Math. Soc.

FORMALIZING THE META-THEORY OF FIRST-ORDER PREDICATE LOGIC

Hugo Herberlin, SunYoung Kim, and Gyesik Lee

Abstract. This paper introduces a representation style of variable binding using dependent types when formalizing meta-theoretic properties. The style we present is a variation of the Coquand-McKinna-Pollack's locally-named representation. The main characteristic is the use of dependent families in defining expressions such as terms and formulas. In this manner, we can handle many syntactic elements, among which well-formedness, provability, soundness, and completeness are critical, in a compact manner. Another point of our paper is to investigate the roles of free variables and constants. Our idea is that fresh constants can entirely play the role of free variables in formalizing meta-theories of first-order predicate logic. In order to show the feasibility of our idea, we formalized the soundness and completeness of LJT with respect to Kripke semantics using the proof assistant Coq, where LJT is the intuitionistic first-order predicate calculus. The proof assistant Coq supports all the functionalities we need: intentional type theory, dependent types, inductive families, and simultaneous substitution.

1. Introduction

In predicate logic, two sorts of variable binding are involved. The binding of bound variables is used for representing universal quantification, such as

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

and the binding of free variables is used for representing parametric derivations, such as

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

In traditional mathematical usage, it is very common to use the same set of variables for both sorts of binding. However, this common practice turned out to be not so practical in a mechanical development of a formal meta-theory. The main issue with this approach is that bound variables sometimes clash with free variables. For instance, the standard definition of unrestricted substitution for the Lambda calculus by Curry [9, p. 94] can cause the occurrence of variable capture during substitution, and many proofs involving substitution become notoriously tedious because one cannot define substitution by a

2000 *Mathematics Subject Classification.* 03F03, 03F05, 03F30.

Key words and phrases. Formal proofs, first-order predicate logic, Kripke semantics, soundness, completeness.

structural induction due to the danger of variable capture. A typical way of addressing this issue is to work with α-conversion.

However, it turned out that dealing with α-conversion in a formal way is not so feasible either, because this requires a huge amount of extra work. One exceptional example is given by nominal techniques in Isabelle/HOL [27] based on the nominal logic introduced by Pitts et al. in [10, 23]. However, to the best of our knowledge, there exists no user-friendly work dealing with α-conversion formalized in an intentional proof assistant, such as Coq.

Coquand [8] recognized that one can apply the idea of distinguishing between the two sorts of variables in order to avoid to reason about α-conversion. Following his suggestion, McKinna and Pollack extensively investigated the main characteristics involved in employing two sorts of variables in formally proving meta-theories of Lambda calculus and Pure Type Systems [19, 20]. In particular, they showed that many important properties of a typed lambda calculus can be stated and proved without referring to α-conversion, such as Church-Rosser, standardization, and subject reduction. We call the technique developed in [8, 19, 20] *the Coquand-McKinna-Pollack style locally-named representation*.

The first contribution of this paper is to present a variation of the Coquand-McKinna-Pollack style locally-named representation. The variation makes use of dependent type programming in representing the language syntax. The core idea is to define expressions like terms and formulas as dependent families. For a list of variables m, `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 a *trace*, collects the bound variables that can possibly occur *unbound* in a term or a formula although they are supposed to be bound by a $\forall$-quantifier.[1] Furthermore, the family `term` *nil* (resp. `formula` *nil*) denotes the set of all well-formed terms (resp. formulas). Here, *nil* denotes the empty list. Using this idea, one can for example give a more natural representation of the derivability predicate without additional reference to well-formedness. This is a major difference from the style of dealing with well-formedness in the study of McKinna and Pollack [19, 20]. We provide a full explanation of how this idea is applied to the formalization of the meta-theory of the intuitionistic first-order predicate logic.

In this paper we also investigate the role of free variables. During our formalization work, we found out that free variables essentially play no syntactic role, except the case where they are replaced by bound variables in stating the generality of judgments. This fact can be seen in the following two rules with respect to $\forall$-quantification where variable binding occurs:

[1] 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 sets of variables.

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

These are two rules in Gentzen-style sequent calculi representing the left and right introduction rules of $\forall$-quantification, respectively. Furthermore, these are the only rules in which instantiation of a bound variable occurs. Note that instantiation of the $\forall$-quantifier by an arbitrary term occurs only in $(\forall_L)$ while in $(\forall_R)$ a bound variable is instantiated by a free variable. As we will see in Figure 3 later which illustrates the whole presentation of an intuitionistic Gentzen-style sequent calculus, called the cut-free LJT, it is not necessary to consider instantiation of free variables in the definition of the deduction system. This observation gives rise to the question of whether we need free variables at all when we formalize meta-theories of a logical system. This drove us to investigate whether we could show the same meta-theoretic results even when we do not incorporate free variables into the language. We show that one can instead let fresh constants play the role of free variables.

We employ the proof assistant Coq [7] as the programming tool. Coq provides all the functionalities we need to realize our ideas: intentional type theory, dependent types, inductive families, and simultaneous substitution.[2]

The rest of the paper is organized as follows. Section 2 describes the syntactic part of an intuitionistic predicate calculus LJT and discusses technical details regarding our formalization, such as 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 for LJT such as soundness, completeness, and cut-admissibility. Section 4 concludes with a summary and some remarks.

2. Presentation of the intuitionistic sequent calculus LJT

It is well known that nominal representation using one sort of variable is not feasible for formal proofs in an intentional theorem prover when variable binding is involved. There have been some trials that demonstrate the feasibility of the nominal representation style, such as Stump's partial contribution to the POPLmark Challenge[25]. However, on a larger and more complicated scale, the notorious problem with variable capture has remained unsolved. On the other hand, when one works with an intentional proof assistant, the locally-named representation [19, 20] and the locally nameless representation [2, 6] are excellent choices. Each style has advantages and disadvantages, but our interest lies in the locally-named representation, because of its use of named variables for binding.

Our work began with the observation that both representations are too permissive in the definition of terms. Some terms cannot have any meaning, because they contain free occurrences of some bound variables that are subject

[2]The Coq scripts are available at `https://github.com/liganega/trace`.

to be bound. This gives rise to the necessity of an extra syntax for the so-called *well-formed* terms. It is a distinctive feature of McKinna and Pollack's work that they introduced a method of avoiding the appeal to such well-formedness considerations, except for a very small number of places such as the definition of typing rules.

Remark 2.1. There are approaches with two sorts of variables that require no extra syntax for well-formedness. For example, Sato and Pollack [24] introduced the so-called *internal syntax*, where all expressions are well-formed although two sorts of named variables are used. In fact, no α-conversion is necessary, because all expressions are unique themselves, as this is the case with the approach based on de Bruijn indices. However, we do not consider such approaches, because the mechanism they employ is not related to our interest.

In order to remove the necessity of employing extra syntax for well-formed terms, we use *traces* to control information regarding bound variables occurring in the construction of terms. The idea is as follows. The elements of a trace are not relevant, per se, which is reflected by the fact that trace relocation has no impact on substitution and Kripke semantics. The only important point is their occurrence in the trace, which is tracked by proofs of the membership. This allows names and de Bruijn indices to be superimposed. Indeed, their relationship can be observed when we examine the use of de Bruijn indices in McBride and McKinna's typechecker example from [18, 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 $x \in y :: m$.

Our idea is also very close to the use of families in Agda, which is indexed by Nat to represent well-scoped terms. The origin of the idea appears to go back to [5, 4, 1].

2.1. Predicate language without free variables

As explained in the introduction, free variables appear to play no essential role for the establishment of meta-theories of first-order predicate logic. We also wish to verify this, and apply our idea to the formalization of LJT, which is a Kripke-based semantical cut-elimination of an intuitionistic first-order predicate logic. We employ a version of the locally-named representation. A main characteristic of the language of LJT is that it contains bound variables, but no free variables.

The language of LJT involves two kinds of expressions, namely terms and formulas. The definition of formulas involves universal quantification $\forall$, which is a kind of variable binding. Therefore, we believe that this provides an appropriate case study for testing and confirming our ideas.

We adopt sequent calculus style derivability to represent proofs. The advantage of such an approach is that it involves an easy-to-define notion of the

Terms:

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

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

Formulas:

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

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

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

Occurrence of variables:

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

Occurrence of constants:

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

FIGURE 1. Terms and formulas without free variables

normal form. A proof is in normal form when it is merely constructed without using the cut rule.

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

We use names to represent both bound variables and constants. Letters such as c, d, c_i, d_i vary over constants while letters such as 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 of the sets mentioned here are assumed to be *decidable*. A set X is *decidable* if, constructively, $\forall u, v \in X\,(u = v \vee u \neq v)$ holds. That is, 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. For instance, the list $x_1 :: \cdots :: x_n :: nil$ of variables denotes the set $\{x_1, ..., x_n\}$, where the order of variable occurrences is important. 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 employ the usual set-theoretic notations such as $\in$, $\notin$, and $\subseteq$.

As mentioned before, one of our main ideas is to define terms and formulas as dependent families. Given a list m of variables, the type $\mathtt{term}\,m$ (resp. $\mathtt{formula}\,m$) denotes the set of terms (resp. formulas), where bound variables from m can possibly occur unbound. The basic notions are explained in Figure 1.

The crucial element in the definition of $\mathtt{Var}\,x\,h \in \mathtt{term}\,m$ is the condition $(h : x \in m)$ which means that h is a witness that x is contained in the trace m. 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 should be contained in the trace.

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

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

2.2. Substitution and trace relocation

We pay special attention to the definition of the substitution. There are two reasons for this. First, in order to establish the soundness and completeness of LJT with respect to a Kripke semantics in a natural manner, it is necessary to work with a simultaneous substitution.

Second, 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 $s \in \mathtt{term}\ m'$. Then, 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 define substitution such that it respects the following two points:

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

The idea is as follows. First, we declare a trace ℓ of variables that are prohibited from being substituted, and then substitute only well-formed terms.

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

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

$$(\dagger) \quad [\ell \Uparrow \eta]\,(\mathtt{Var}\, y\, h) = \begin{cases} \mathtt{Var}\, y\, h' & \textit{if } y \in \ell \\ \mathtt{treloc}\, u_j\, h_j & \textit{if } y \notin \ell \textit{ and } j = \min\{i : y = x_i\} \\ \mathtt{Cst}\, 0 & \textit{otherwise} \end{cases}$$

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

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

Here

- h' is the proof of $y \in \ell$,
- 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:

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

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

$$(\ddagger) \quad [\ell \Uparrow \eta]\,(\forall x\, B) = \forall x\,([x :: \ell \Uparrow \eta]\, B)$$

FIGURE 2. Simultaneous substitution for terms and formulas

The role of ℓ is well demonstrated in the abstraction case ($\ddagger$), where the trace is extended by a bound variable x in order to forbid any substitution for x.

The point is that we know where the resulting term will arrive before the substitution is performed. 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 forces us to work with more intuitive definitions and proofs, such as the inference rules in Figure 3 and the universal completeness in Theorem 3.6. A demonstration of how the substitution works will be given in Example 2.6.

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

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

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

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

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

Two points should be mentioned regarding the definition. First, some variables are ignored by assigning $\mathtt{Cst}\,0$ as in (†). This does not cause any problem because in our work all free occurrences of variables should be covered either by the list ℓ or by the domain of an association.

Remark 2.4. In the formal definition of substitution in Coq, the ignored case could be handled in a different manner. Namely, by including appropriate extra propositional arguments denoting the side condition that $\mathtt{OV}(e) \subseteq \mathsf{dom}(\eta)$ or $\mathtt{OV}(A) \subseteq \mathsf{dom}(\eta)$. This would make our work more perfect for application to dependently typed programming. However, the definition given above works more smoothly from a technical point of view.

Second, we have to employ trace relocation in (†). The substituted term u_j is of type $\mathtt{term}\,nil$. In order for typechecking to work, we need to relocate this to $\mathtt{term}\,\ell$, and this is the reason why trace relocation is required.

In the following, we simplify our notation 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 $\mathtt{OV}(t) \subseteq \ell$:

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

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

The relocation function is homomorphic in the sense that it does not change or disrupt the functionality of any terms, either syntactically or semantically. Note also that the proof element in the definition of a term is inessential in the case that the trace contains all necessary variables and that $\mathtt{treloc}(t)$ does not alter anything but proof part elements. Indeed, one can show that relocation has no impact on substitution.

Lemma 2.5. *Let ℓ be a trace, t a term, and η an association. Then we have*

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

To demonstrate the effectiveness of our definition of substitution we give an example.

Example 2.6. *In the language of Peano arithmetic (PA), consider the following formula*

$$A(x,y) \equiv x < y \to \forall z(x + z < y + z)\,.$$

Note that the variables x and y are not bound. Therefore $A(x,y)$ is not well-formed and belongs to $\mathtt{formula}\,\ell$, where $\ell = y :: x :: nil$. On the other hand, $\forall x\,y\,A(x,y) \in \mathtt{formula}\,nil$, hence a well-formed formula which is provable in PA. From this fact, it follows that the formula $A(1,2)$ should also be provable in PA. However, before we talk about its provability, we have to first guarantee its well-formedness. In fact, the inference rules in Figure 3 involve only well-formed formulas, i.e., formulas from the set $\mathtt{formula}\,nil$.

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

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

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

FIGURE 3. Cut-free LJT

Informally, people used to assume the fact that $\mathtt{OV}(A(1,2)) = \varnothing$. *However, in our work, this kind of assumption is not necessary at all because we have by definition*

$$(*) \qquad [\,nil \Uparrow \eta\,]\,A(x,y) \in \mathtt{formula}\, nil,$$

where $\eta = (x,1),(y,2)$. *The well-formedness of* $A(1,2)$ *follows directly from* $(*)$ *because it is just the result of the substitution:*

$$\begin{aligned}
[\,nil \Uparrow \eta\,]\,A(x,y) &= [\,nil \Uparrow \eta\,]\,(x < y \to \forall z(x + z < y + z)) \\
&= [\,nil \Uparrow \eta\,]\,(x < y) \to [\,nil \Uparrow \eta\,]\,(\forall z(x + z < y + z)) \\
&= 1 < 2 \to \forall z([\,z :: nil \Uparrow \eta\,]\,(x + z < y + z)) \\
&= 1 < 2 \to \forall z(1 + z < 2 + z)\,.
\end{aligned}$$

In addition to the example above, we also stress that the substitution lemma can be proved relatively easily just by using a structural induction. The fact that it is usually not the case is well explained e.g. in [3].

Lemma 2.7 (Substitution Lemma)**.** *Let* ℓ *be a trace,* e *an expression,* $u \in \mathtt{term}\, nil$, *and* η *an association. Then,*

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

2.3. 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 the *stoup* and contains the principal formula of the corresponding left introduction rule.

The formal definition in Coq of the inference rules can be represented exactly as in Figure 3 without including any side condition, because a context is of type $\mathtt{list}\,(\mathtt{formula}\, nil)$ and a well-formed formula is of type $\mathtt{formula}\, nil$.

Remark 2.8. Herbelin [12, 13] and Mints [21] showed that cut-elimination matches normalization in the $\bar{\lambda}$-calculus, which is a variant of λ-calculus for the sequent calculus structure. This implies that LJT effectively supports the proofs-as-programs correspondence.

The right quantification rule ($\forall_R$) requires some explanation. Note first that a fresh constant c is employed in the premise of the rule. This relies on the fact that a fresh constant can be used instead of a fresh free variable.

Next, we must explain our choice of quantification style. It is sufficient for the premise of ($\forall_R$) to hold for *one* fresh constant. There are some issues regarding this style of quantification, such as the fact that it provides too weak an induction principle. For example, let us attempt 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, then 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 attempt to prove this would lead to weakening, a vicious circle. An excellent solution for breaking this circle is provided by Pitts [23]. He showed that, by employing swapping, one can easily prove renaming without appealing to weakening.

Here we explain another option that enables us to prove weakening and renaming simultaneously. Proving weakening and renaming simultaneously appears to be a natural idea, because 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. In the following, $\rho = (c_1, d_1), ..., (c_n, d_n)$ denotes a simultaneous renaming of constants. Given a formula $A \in \mathtt{formula}\, m$, the formula $\rho\, A$ is of type $\mathtt{formula}\, m$, 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 using a simple, structural induction. Weakening and renaming are special forms of this theorem.

Theorem 2.9 (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 hold:*

(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. We just remark that this is because derivability is predicate and does not belong to the part of the domain of the discourse. Otherwise,

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

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

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

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

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

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

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

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

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

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

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

FIGURE 4. Kripke semantics

some kind of bijectivity of the renaming will be necessary as demonstrated by McKinna and Pollack [20].

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

Having seen the basic syntax of LJT, in this section we provide a Kripke semantics for LJT. Kripke semantics was created in the late 1950s and early 1960s by Saul Kripke in [16, 17]. It was first introduced for modal logic, and later adapted to intuitionistic logic and other non-classical and classical systems in (cf. [26, 15]). Here, we employ the conventional Kripke model adopted by Troelstra and van Dalen.

A Kripke model $\mathcal{K} = (\mathcal{W}, \leq, \Vdash, \mathcal{D}, V)$ is a tuple of a partially-ordered set $\mathcal{W}$ of *worlds*; a domain $\mathcal{D}$; interpretations of constant and function symbols into

the domain; and a relation between worlds, predicates, and domain elements (cf. Figure 4). The interpretation of terms is based on an association η whose codomain is $\mathcal{D}$. Note that some variables are ignored. This is necessary to cope with the definition of simultaneous substitution, where some variables are also ignored. Furthermore, the proof term for a list membership is simply neglected, such that the trace relocation has no impact on the Kripke semantics.

Soundness and completeness can be formalized without any difficulty.

Theorem 3.1 (Soundness)**.** *One can prove the following simultaneously.*

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

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

If we had included free variables and let them play their intended role, then the soundness proof would be very simple to prove, as shown in [14]. However, because constants take the role of free variables, the $(\forall_R)$ rule requires more attention.

Suppose $\Gamma \vdash \forall x\, A$ 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 $d \in \mathcal{D}$, we have to show that

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

holds. At this point, the premise of $(\forall_R)$ appears 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 free variables. 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* $\mathcal{K} = (\mathcal{W}, \leq, \Vdash, \mathcal{D}, V)$*, a constant* c*, and a value* $d \in \mathcal{D}$*, we define a new Kripke model* $\mathcal{K}_{c,d} := (\mathcal{W}, \leq, \Vdash, \mathcal{D}, V_{c,d})$*, where*

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

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

Lemma 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* $\mathcal{K} = (\mathcal{W}, \leq, \Vdash, \mathcal{D}, V)$*, any* $w \in \mathcal{W}$*, and any* $d \in \mathcal{D}$*, we have*

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

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

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

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

where the first equivalence 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 $\mathcal{K} = (\mathcal{W}, \leq, \Vdash, \mathcal{D}, V)$, any $w \in \mathcal{W}$, and any association η, we have*

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

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

Formalization of completeness is performed in the same manner as in [14]. That is, we use the fact that LJT is complete with respect to a universal Kripke model $\mathcal{U}$ defined as follows:

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

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

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

Note that in the universal model $\mathcal{U}$, the interpretation of terms corresponds to substitution. That is, given a term $t \in \mathtt{term}\, m$ and an association $\eta = (x_1, u_1), ..., (x_n, u_n)$, where $u_i \in \mathtt{term}\, nil$, we have $t[\eta] = [\, nil \,\Uparrow\, \eta \,]\, t$. The universal completeness, as stated below, implies that we have a similar correspondence between forcing and deduction.

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

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

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

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, so do $\Gamma \Vdash_{\mathcal{U}} A$ and $\Gamma \Vdash_{\mathcal{U}} B$. Consequently, $\Gamma \vdash B$ holds by the universal completeness. □

Remark 3.9. Because (Cut) is a semantically sound rule, a composition of (soundness) and (universal completeness) normalizes any proof with (Cut) to a cut-free proof. A program extraction (which is available in Coq) from the composition would provide a functional program that produces a cut-free proof from a deduction with (Cut). We believe that the normalization follows the reduction semantic of LJT.

4. Conclusion

The main idea of this paper is that the Coquand-McKinna-Pollack style locally-named representation can be successfully employed in the formalization of a logical meta-theory with variable binding, especially when the proofs-as-programs correspondence is irrelevant, which is usually the case for logicians and mathematicians.

Moreover, our work uses traces to have control over variables used in terms or formulas. With traces one can comfortably work with syntax, such as well-formedness and provability. The elements of a trace are not relevant, per se, which is reflected by the fact that trace relocation has no impact on substitution and Kripke semantics. The only important point is their occurrence in the trace, which is tracked by proofs of list membership. This allows names and de Bruijn indices to be superimposed.

However, working with traces requires dependently typed programming. In the case of Coq, working with dependent types is sometimes heavy-going. This is one reason why simultaneous substitution is defined as a kind of partial function. We believe that one might have it easier with other tools such as Agda [22].

Acknowledgement

The second author was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea(MSIP) (No. NRF-2013R1A1A2073702, NRF-2017R1C1B1004836), and by the Yonsei University Research Fund(Post Doc. Researcher Supporting Program) of 2016 (project no.: 2016-12-0014). The third author was partially supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MOE) (No. NRF-2017R1D1A1B05031658).

References

[1] T. Altenkirch and B. Reus, *Monadic presentation of lambda terms using generalized inductive types*, Lecture Notes in Computer Science, **1683** (1999), 453–468.

[2] B. Aydemir, A. Charguéraud, B. C. Pierce, R. Pollack, and S. Weirich, *Engineering formal metatheory*, ACM SIGPLAN Notices, **43** (2008), no. 1, 3–15.
[3] S. Berghofer and C. Urban, *A Head-to-Head Comparison of de Bruijn Indices and Names*, Electronic Notes in Theoretical Computer Science, **174** (2007), no. 5, 53–67.
[4] R. S. Bird and L. Meertens, *Nested datatypes*, Lecture Notes in Computer Science, **1422** (1998), 52–67.
[5] R. S. Bird and R. Paterson, *De Bruijn notation as a nested datatype*, Journal of Functional Programming, **9** (1999), no. 1, 77–91.
[6] A. Charguéraud, *The locally nameless representation*, Journal of Automated Reasoning, **49** (2012), no. 3, 363–408.
[7] Coq Development Team, *The Coq Proof Assistant Reference Manual*, Available at `http://coq.inria.fr`.
[8] T. Coquand, *An algorithm for testing conversion in Type Theory*, in G. Huet and G. Plotkin, editors, *Logical Frameworks*, Cambridge University Press, 1991, 255–279.
[9] H. B. Curry and R. Feys, *Combinatory Logic. Volume 1*, North Holland, 1958.
[10] M. J. Gabbay and A. M. Pitts, *A new approach to abstract syntax with variable binding*, Formal Aspects of Computing, **13** (2002), no. 3-5, 341–363.
[11] G. Gentzen, *Untersuchungen über das logische Schließen. I*, Mathematische Zeitschrift, **39** (1934), no. 2, 176–210.
[12] H. Herbelin, *A λ-calculus structure isomorphic to gentzen-style sequent calculus structure*, Lecture Notes in Computer Science, **933** (1994), 61–75.
[13] H. Herbelin, *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.
[14] H. Herbelin and G. Lee, *Forcing-based cut-elimination for gentzen-style intuitionistic sequent calculus*, Lecture Notes in Computer Science, **5514** (2009), 209–217.
[15] D. Ilik, G. Lee, and H. Herbelin, *Kripke models for classical logic*, Ann. Pure Appl. Logic, **161** (2010), no. 11, 1367–1378.
[16] S. Kripke, *A Completeness Theorem in Modal Logic*, Journal of Symbolic Logic, **24** (1959), no. 1, 1–14.
[17] ——, *Semantical considerations on modal and intuitionistic logic*, Acta Philosophica Fennica, **16** (1963), 83–94.
[18] C. McBride and J. McKinna, *The view from the left*, Journal of Functional Programming, **14** (2004), no. 1, 69–111.
[19] J. McKinna and R. Pollack, *Pure type systems formalized*, Lecture Notes in Computer Science, **664** (1993), 289–305.
[20] J. McKinna and R. Pollack, *Some lambda calculus and type theory formalized*, Journal of Automated Reasoning, **23** (1999), no. 3-4, 373–409.

[21] G. Mints, *Normal forms for sequent derivations*, in P. Odifreddi, editor, *Kreiseliana*, A. K. Peters, Wellesley, 1996, 469–492.
[22] U. Norell, *Dependently typed programming in Agda*, in A. Kennedy and A. Ahmed, editors, *Proceeding of TLDI'09*, ACM, 2009, 1–2
[23] A. M. Pitts, *Nominal logic, a first order theory of names and binding*, Information and Computation, **186** (2003), no. 2, 165–193.
[24] M. Sato and R. Pollack, *External and internal syntax of the lambda-calculus*, Journal of Symbolic Computation, **45** (2010), no. 5, 598–616.
[25] A. Stump, *Poplmark 1a with named bound variables*. Available at `https://www.seas.upenn.edu/~plclub/poplmark/stump.html/`.
[26] A. S. Troelstra and D. van Dalen, *Constructivism in Mathematics: An Introduction I and II*, vol. 121, 123, North-Holland, 1988.
[27] C. Urban, *Nominal Techniques in Isabelle/HOL*, Journal of Automated Reasoning, **40** (2008), no. 4, 327–356.

Hugo Herberlin
Laboratoire IRIF-PPS
INRIA, PPS
75205 Paris Cedex, France
E-mail address: `Hugo.Herberlin@inria.fr`

SunYoung Kim
Department of Mathematics
Yonsei University
Seoul 03722, Korea
E-mail address: `sunyoungkim831@gmail.com`

Gyesik Lee
Department of Computer Science and Engineering
Hankyong National University
Anseong 17579, Korea
E-mail address: `gslee@hknu.ac.kr`