Journal of Engineering and Applied Sciences 13 (19): 7878-7883, 2018
ISSN: 1816-949X

Locally-Named versus Nominal

[1]Hwajeong Kim and [2]Gyesik Lee
[1]Department of Mathematics, Hannam University, Expo-ro 448, Yuseong-gu,
34049 Daejeon, Korea
[2]Department of Computer Science and Engineering, Hankyong National University,
Gwangpyeong-ro 10-Gil 6, Gangnam-gu, 16359 Seoul, Korea

Abstract: The main characteristic of our representation is the use of dependent families in defining expressions such as terms and formulas. Another point is that we do not consider parameters and show that we can do the same thing as when parameters are involved. In order to confirm the feasibility of our idea we made several experiments using the proof assistant Coq.

Key words: Mechanical formalization, variable binding, proof assistants, dependent families, parameters, experiments

INTRODUCTION

Around the turn of the 20th century, mathematicians and logicians were interested in a more exact investigation into the foundation of mathematics and soon realized that ordinary mathematical arguments can be represented in formal axiomatic systems. A formal approach was initiated by Frege (1879). He invented in Begriffsschrift (Frege, 1879) a special kind of language system where statements can be proved as true based only upon some general logical laws and definitions. His concern was about applying pure logic to arithmetic judgments.

His research is known to be inconsistent but addresses all the three types of concern that can attend a mathematical proof which are mentioned in (Avigad and Harrison, 2014) mathematical appropriateness of methods, correct use of appropriate methods and appropriate understanding.

Indeed, Frege said the following "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 (Ebert and Rossberg, 2013)".

A main characteristic of Frege's approach is that 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.

In this study, we focus on an aspect of the tradition of applying a rigorous scientific method for mathematics, namely formal proof and give an overview of an approach to formally proving meta-theories of first-order predicate logic. A formal proof is a proof which is written in an artificial language. And in the present-day practice, 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.

Our main claim is that when the Coquand-McKinna-Pollack style locally-named representation is used without parameters it results in a nominal representation. Another aspect of our work can be found by Herbelin and Lee (2009).

MATERIALS AND METHODS

Motivation: In predicate logic, two sorts of variable binding are involved:

- Binding local variables is used for representing universal quantification as in $\vdash \forall x\, P(x)$
- 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 sorts of variables. However, this common practice is not so practical in a mechanical development of a formal meta-theory. A typical way of addressing this issue is to work with α-conversion.

However, dealing with α-conversion formally is not so, feasible. People, however, recognized that Frege's idea of distinguishing between the two sorts of variables can

Corresponding Author: Gyesik Lee, Department of Computer Science and Engineering, Hankyong National University, Gwangpyeong-ro 10-Gil 6, Gangnam-gu, 16359 Seoul, Korea

be practically applied in doing machine-checked proofs. Coquand suggested by Coquand (1991) to use Frege's idea to avoid the need to reason about α-conversion. Following his suggestion, McKinna and Pollack (1993, 1999) extensively investigated the main characteristics of using two sorts of variables in proving the meta-theories of lambda calculus and pure type systems.

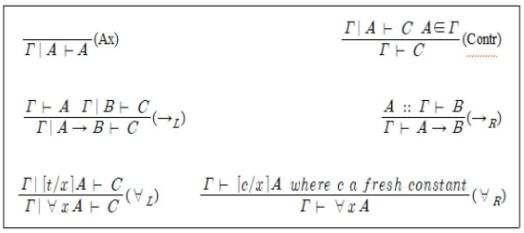
Another solution can be given when we observe that parameters syntactically play no essential role in Frege's work except when they are replaced by local variables in stating the generality of judgments. This fact can be for instance, observed in the left and right introduction rules of universal quantification. And they are the only rules where instantiation of a bound local variable occurs. Indeed it is not necessary to consider instantiation of parameters in the definition of the deduction system.

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 at all.

Formal presentation of LJT without parameters: The main contribution of our work is to give an answer to the question explained in the motivation. Indeed, we have confirmed the feasibility of using no parameters and letting constants play the role of parameters in formalizing logical meta-theory. We remark that our idea goes through 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 represented below in Fig. 1. denotes a context which is a finite set of sentences. The corresponding definition in Coq is presented as follows:

From the logical point of view, what we have done is very similar to what (Coquand, 1993, 2002) 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. In our case, although a version of the substitution lemma is still necessary in order to prove the completeness of LJT, our work does not involve any instantiation of parameters.

Another contribution of our research is that we provide a reasonable application of dependent type programming in representing language syntax of a predicate logic as in Fig. 2. The core of our idea lies in the definition of terms and formulas. They are defined by dependent families.



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

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

$$\frac{\Gamma \mid [t/x]A \vdash C}{\Gamma \mid \forall x A \vdash C}(\forall_L) \qquad \frac{\Gamma \vdash [c/x]A \quad \text{where } c \text{ a fresh constant}}{\Gamma \vdash \forall x A}(\forall_R)$$

Fig. 1: LJT

```
Inductive prove : context -> fml -> Type :=
  | ProofCont : forall (A C: fml) (Ga : context),
    IN_ctx A Ga -> Ga ;; A |- C -> Ga |- C

  | ProofImplyR : forall B C Ga,
    B :: Ga |- C -> Ga |- B --> C

  | ProofForallR : forall y (B : fml) Ga (a : name),
    a # oc_c (B :: Ga) ->
    Ga |- open B y (Cst a) ->
    Ga |- (Forall y B)

  where "Ga |- A" := (prove Ga A)

with prove_stoup : context -> fml -> fml -> Type :=
  | ProofAxiom Ga C: wf_c Ga -> wf C -> Ga ;; C |- C

  | ProofImplyL Ga D : forall B C,
    Ga |- B -> Ga ;; C |- D -> Ga ;; (B --> C) |- D

  | ProofForallL Ga C : forall y (u : trm) (B : fml),
    wf_t u ->
    Ga ;; open B y u |- C ->
    Ga ;; Forall y B |- C

  where "Ga ;; A |- C" := (prove_stoup Ga A C).
```

Fig. 2: LJT in Coq

Terms: Let m denote a set of bound variables. Then the inductive familily **term** m is defined as follows:

- **Var** $x\,h \in$ **term** m, where x is a bound variable and h is a proof that $x \in m$.
- **Cst** $c \in$ **term** m, where c is a constant symbol.
- **App** $f\,t_1\,t_2 \in$ **term** m, where f is a function symbol and $t_1, t_2 \in$ **term** m.

Formulas: Let m denote a set of bound variables. Then the inductive familily **formula** m is defined as follows:

- **Atom** $(P,t) \in$ **formula** m, where P is a predicate symbol and $t \in$ **term** m.
- $A \to B \in$ **formula** m, where $A, B \in$ **formula** m.
- $\forall x A \in$ **formula** m, where x is a bound variable and $A \in$ **formula** $(x :: m)$.

Contexts: **context** is the set of all finite sets of sentences, i.e., closed formulas.

Fig. 3: Dependent type representation of terms and formulas without parameters

More concretely, 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. In this way, one can give a more natural representation of, e.g., the derivability predicate (Fig. 3 and 4). The corresponding definitions of term and formula families looks as follows:

```
Inductive trm : Set :=
| BVar : name -> trm
| Cst  : name -> trm
| App  : function -> trm -> trm -> trm.

Definition atom := (predicate * trm)%type.

Inductive fml : Set :=
| Atom   : atom -> fml
| Imply  : fml -> fml -> fml
| Forall : name -> fml -> fml.
```

Fig. 4: Terms and formulas in Coq

RESULTS AND DISCUSSION

Formal presentation of Kripke semantics: Two important meta-theoretic properties of a logical system are its soundness and completeness with respect to a semantics. For that purpose, we used a Kripke style semantics for LJT. Kripke semantics was created in the late 1950's and early 1960's by Kripke (1959, 1963).

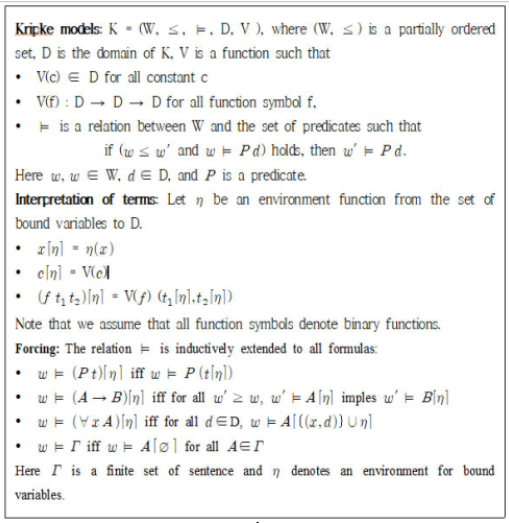
Kripke model was first introduced for modal logic and later adapted to intuitionistic logic and other non-classical or classical systems (Troelstra and Dalen, 1988; Ilik *et al*., 2010). Here, we use the conventional Kripke model adopted by Troelstra and van Dalen (1988).

Definition (Kripke Model): A Kripke model K = (W, ≤, ⊨, D, V) is a tuple of a partially-ordered set W, a domain D interpretations of constant and function symbols into the domain and a relation between worlds, predicates and domain elements (Fig. 5). Interpretation of terms is based on an environment η. The corresponding definition of Kripke semantics looks as follows:

Algorithm; Kripke Model:

```
Fixpoint psem K (t: term) (eta:assocs (domain K)) {struct t}:
domain K: =
match t with
| Bvar x = ⟩ match lookup eta x with
               | Some u = ⟩ u
               | None = ⟩ csts_θ (*fixed value*)
             end
| Cst c = ⟩ csts_c
| App f t1 t2  = ⟩ funs_f  (psem  k  t1  eta) (psem k t2 eta)
end.
Fixpoint force K (w: worlds K) (A : fml) eta  {struct A} : =
match A with
    | Atom (p, t)  = ⟩ atoms K w (p,  psem  K t eta)
    | B --⟩ C        = ⟩  forall  w′ , w ⟨= w′ -⟩
                                    w′ | |- B {[eta]}-⟩ w′ | |- C{[eta]}
|Forall y B =⟩ forall  w′ , w ⟨= w′ -⟩
                forall (d : domain K),  w′ | |- B {[(y,d)  :: eta]}
end

where ″ w | |- A {[eta]}  ″ := (force_w  A eta)
```



Kripke models: K = (W, ≤, ⊨, D, V), where (W, ≤) is a partially ordered set, D is the domain of K, V is a function such that

- V(c) ∈ D for all constant c
- V(f) : D → D → D for all function symbol f,
- ⊨ is a relation between W and the set of predicates such that if $(w \leq w'$ and $w \vDash P\,d)$ holds, then $w' \vDash P\,d$.

Here $w, w \in W$, $d \in D$, and P is a predicate.

Interpretation of terms: Let η be an environment function from the set of bound variables to D.

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

Note that we assume that all function symbols denote binary functions.

Forcing: The relation ⊨ is inductively extended to all formulas:

- $w \vDash (P\,t)[\eta]$ iff $w \vDash P\,(t[\eta])$
- $w \vDash (A \to B)[\eta]$ iff for all $w' \geq w$, $w' \vDash A[\eta]$ implies $w' \vDash B[\eta]$
- $w \vDash (\forall x\,A)[\eta]$ iff for all $d \in D$, $w \vDash A[((x,d)) \cup \eta]$
- $w \vDash \Gamma$ iff $w \vDash A[\varnothing]$ for all $A \in \Gamma$

Here Γ is a finite set of sentence and η denotes an environment for bound variables.

Fig. 5: Kripke semantics

Soundness and completeness can be formalized without any difficulty.

Theorem (Soundness): (1) Let be a set of sentences and C a sentence. Suppose ⊢ C holds. For any Kripke model K = (W, ≤, ⊨, D, V) and any w ∈ W, if w ⊨ holds, so does w ⊨ C. (2) Let be a set of sentences and A, C sentences. Suppose |A⊢ C holds. For any Kripke model K = (W, ≤, ⊨, D, V) and any w ∈ W, if w ⊨ and w ⊨ A hold, so does w ⊨ C.

Formalization of completeness proof is done in the same way as by Herbelin and Lee (2009). That is, we use the fact that LJT is complete with respect to a universal Kripke model U defined as follows:

Definition (universal Kripke Model): A universal Kripke Model is U = (Ξ, ⊆, ⊨, D, V) where:

- Ξ is the set of all contexts
- ⊆ is the subset relation
- D is the set of closed terms
- V is defined as follows: V(c) = c and V(f) (t1, t2) = f t1 t2
- ⊨ P t iff Ã ⊢ P t holds

The following universal completeness says that there is a direct correspondence between semantics and deduction.

Theorem (universal completeness): Let A be a sentence, ¥Ã a context and η an association. Then, ¥Ã ⊨ [⊙] implies ¥Ã ⊢ A. Now, the completeness follows.

Theorem (completeness): Let A be a sentence and ¥Ã a context. If for any Kripke model K = (W, ≤, ⊨, V) and any w ∈ W, w ⊨ A follows from w ⊨ ¥Ã, then we have ¥Ã ⊢ A.

Locally-named vs. nominal: This section explains another contribution of our work. 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 term m and 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 quantification 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:

- Term m denotes the family of terms with possible occurrences of parameters from the finite set m
- Formula m denotes the family of formulas with possible occurrences of parameters from the finite set m
- Term ⊙ denotes the family of closed terms, i.e., no occurrences of parameters
- Formula ⊙ denotes the family of sentences, i.e., closed formulas

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 study.

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 study.

Practical formal meta-theory: 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:

- The de Bruijn index 0 corresponds to a proof that 0 ∈ { x } ∪ m
- The de Bruijn successor S on indices corresponds to the proof that x ∈ m implies x ∈ 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. We have not tested yet but it could work smoothly with other tools such as Agda.

In order to check the plausibility of our ideas introduced in this study, 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 study, 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 study.

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 and slimed it down to handle the simply-typed lambda calculus, still using the locally nameless representation. The main changes we made are as follows. First, simultaneous substitution instead of single substitution: we checked that simultaneous substitution works well also with de Bruijn indices. Second, 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 by McKinna and Pollack (1999).

CONCLUSION

We used the proof assistant Coq (2) as the programming tool for formalization. The proof assistant Coq provides all the functionalities we need in order to realize our ideas: intentional type theory, dependent types inductive families and simultaneous substitution. 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 renaming, quantification style, comparison between simultaneous renaming and swapping variables. There are several issues to be discussed about what we have accomplished Lee *et al.* (2016).

ACKNOWLEDGEMENTS

For the first research, this research is partly supported by Hannam University in 2018 and by the National Research Foundation of Kore (NRF-2014R1A1A3049709). For the second research, this research is partly supported by the National Research Foundation of Korea (NRF- 2017R1D1A1B05031658).

REFERENCES

Avigad, J. and J. Harrison, 2014. Formally verified mathematics. Commun. ACM., 57: 66-75.

Coquand, C., 1993. From semantics to rules: A machine assisted analysis. Proceedings of the International Workshop on Computer Science Logic, August 24-28, 1993, Springer, Berlin, Germany, ISBN: 978-3- 540-58277-9, pp: 91-105.

Coquand, C., 2002. A formalised proof of the soundness and completeness of a simply typed Lambda-calculus with explicit substitutions. Higher Order Symbolic Comput., 15: 57-90.

Coquand, T., 1991. An Algorithm for Testing Conversion in Type Theory. In: Logical Frameworks, Huet, G. and G. Plotkin (Eds.). Cambridge University Press, Cambridge, UK., ISBN:9780521413008, pp: 255-279.

Ebert, P.A. and M. Rossberg, 2013. Gottlob Frege: Basic Laws of Arithmetic. Oxford University Press, Oxford, UK., ISBN:9780199281749,.

Frege, G., 1879. Begriffsschrift, Eine Der Arithmetischen Nachgebildete Formelsprache Des Reinen Denkens. Hachette Livre Bnf, Paris, France, ISBN-13: 978-2012525719, Pages: 106.

Herbelin, H. and G. Lee, 2009. Forcing-based Cut-elimination for Gentzen-style intuitionistic sequent calculus. Proceedings of the International Workshop on Logic, Language, Information and Computation, June 21-24, 2009, Springer, Berlin, Germany, ISBN:978-3-642-02260-9, pp: 209-217.

Herberlin, H., S. Kim and G. Lee, 2017. Formalizing the Meta-theory of firstorder predicate logic. J. Korean Math. Soc., 54: 1521-1536.

Ilik, D., G. Lee and H. Herbelin, 2010. Kripke models for classical logic. Ann. Pure Applied Logic, 161: 1367-1378.

Kripke, S., 1963. Semantical considerations on modal and intuitionistic logic. Acta Philos. Fennica, 16: 83-94.

Kripke, S.A., 1959. A completeness theorem in modal logic. J. Symbolic Logic, 24: 1-14.

Lee, G., H. Herbelinand S. Kim, 2016. When locally-named becomes nominal: Technical report. Under Consideration Publ. J. Funct. Program., 2016: 1-6.

McKinna, J. and R. Pollack, 1993. Pure type systems formalized. Proceedings of the International Conference on Typed Lambda Calculi and Applications, March 16-18, 1993, Springer, Berlin, Germany, ISBN:978-3-540-56517-8, pp: 289-305.

McKinna, J. and R. Pollack, 1999. Some lambda calculus and type theory formalized. J. Autom. Reasoning, 23: 373-409.

Troelstra, A.S. and V.D. Dalen, 1988. Constructivism in Mathematics: An Introduction. North Holland Publishing Company, New York, USA., ISBN:9780444703583, Pages: 879.