8. C. Paulin-Mohring, "Extracting F_ω's programs from proofs in the calculus of construction", *Proc. 16th Annual ACM Symposium in Principles of Programming Languages* (1989) pp. 89-104.
9. S. Hayashi, "Singleton, Union and Intersection Types for Program Extraction", *Information and Computation* Vol. 109, Nums 1 and 2, (1994) pp. 174-210.
10. Y. Takayama, "Extended projection: a new technique to extract efficient programs from constructive proofs", *Proc. Conference on Functional Programming Languages and Computer Architecture* (ACM Press, 1989).
11. S. Allen, R. L. Constable, D. Howe, and W. Aitken, "The semantics of reflected proof", *Proc. 5th Annual Symposium on Logic in Computer Science* (IEEE Computer Society Press, 1991) pp. 95-105.
12. P. Martin-Löf, *Intuitionistic type theory* (Bibliopolis, 1984).
13. M. J. Beeson, *Foundations of constructive mathematics* (Springer, 1985).
14. P. Aczel, "Frege structures and the notions of proposition, truth, and set", *The Kleene Symposium*, eds. J. Barwise et al. (North-Holland, 1980) pp. 31-59.
15. P. Aczel, D. Carlisle, and N. Mendler, "Two frameworks of theories and their implementation in Isabelle", *Logical Frameworks*, eds. G. Huet and G. Plotkin, (Cambridge University Press, 1991) pp. 3-39.
16. S. Allen, "A non-type-theoretic definition of Martin-Löf's types", *Proc. 2nd Annual Symposium on Logic in Computer Science* (IEEE Computer Society Press, 1987) pp. 215-221.
17. S. Allen, "A non-type-theoretic semantics for type-theoretic language", *Ph. D. Thesis, Cornell University* (1987).
18. S. Smith, "Reflective Semantics of Constructive Type Theory (Preliminary Report)", *LNCS* **613** (Springer, 1991) pp. 33-45.
19. P. Dybjer, "Universes and a general notion of simultaneous inductive-recursive definition in type theory", *Proc. of the 1992 Workshop on Types for Proofs and Programs*, Baastad, eds. B. Nordström et al. (1992).
20. P. Dybjer, "A general notion of simultaneous inductive-recursive definition in type theory", Draft paper (available by ftp from `cs.chalmers.se` at `/pub/clics/peterd/Inductive_Recursive.dvi.Z`) (1993).
21. M. Tatsuta, "Two realizability interpretations of monotone inductive definitions", *International Journal of Foundations of Computer Science* Vol. 5 No. 1 (1994) pp. 1-21.
22. M. Sato, "Adding proof objects and inductive definition mechanisms to Frege structures", *Proc. International Conference on Theoretical Aspects of Computer Software*, eds. T. Ito and A. Meyer, *LNCS* **526** (Springer, 1991) pp. 53-87.
23. D. Park, "Finiteness is Mu-ineffable", *Theoretical Computer Science* Vol 3 (1976) pp. 173-181.
24. P. Dybjer, "Inductive families", *Formal Aspects of Computing*, Vol 6 (1994) pp. 440-465.

range. As a result, we have to confine ourselves to inductive definitions, rather than inductive-recursive definitions. Besides this point, the differences between his work and ours are that (1) his theory is based on type theory while our base theory is untyped. (2) He allows only strictly-positive occurrences for the first predicate variable in his inductive definition, while we allow more general one (monotone inductive definition with **HMONO-Q** condition).

Dybjer[20] also gave a model for his theory. It is an interesting future problem to study the relationship of his model construction and our method.

Concluding Remarks

This work has been done in connection to Sato's $\mathcal{RPT}/\Lambda$ work[22]. $\mathcal{RPT}$ (Reflective Proof Theory) is a type-free constructive theory, but its spirit is very close to Martin-Löf's type theory and Frege structures. It has two judgements $\mathtt{Prop}(p)$ and $\mathtt{Proves}(a,p)$ [d]. If we introduce an **HMID**-like mechanism into $\mathcal{RPT}$, we would be able to define $\langle \mathtt{Prop}, \mathtt{Proves} \rangle$ in $\mathcal{RPT}$, and can extend the theory of program improvement in $\mathcal{RPT}$. We plan to amalgamate our induction mechanism into $\mathcal{RPT}$ smoothly.

Acknowledgements

The author would like to express his heartful thanks to Prof. Masahiko Sato, Prof. Takayasu Ito, and Dr. Carolyn Talcott, Prof. Makoto Tatsuta and Mr. Yasuyuki Tsukada for encouragements, continual supports, and helpful advices. Special thanks go to Dr. Peter Dybjer who provided many insights and pointed out errors in the original draft. The author would like to express his sincere appreciation to anonymous referees who provided many constructive and helpful comments on the original draft.

References

1. S. Feferman, "Constructive theories of functions and classes", *Logic Colloquium '78* eds. M. Boffa et al (North-Holland, 1979) pp. 159-224.
2. R. L. Constable et al., *Implementing mathematics with the Nuprl proof development system* (Prentice-Hall, 1986).
3. T. Coquand and G. Huet, "The calculus of constructions", *Information and Computation*, **76** (1988) pp. 95-120.
4. S. Hayashi and H. Nakano, **PX***: a computational logic* (The MIT Press, 1988).
5. M. Sato and Y. Kameyama, "Constructive programming in $\mathcal{SST}$", *Proc. the Japanese-Czechoslovak Seminar on Theoretical Foundations of Knowledge Information Processing*, Prague (1989) pp. 23–30.
6. B. Nordström, K. Pertersson and J. Smith, *Programming in Martin-Löf's type theory* (Oxford, 1990).
7. S. Hayashi and S. Kobayashi, *Foundations of constructive programming* (in Japanese, "Kouseiteki Puroguramingu no Kiso") (Yusei-sha, 1991)

[d] The original system had an infinite hierarchy of $Prop_i$ and $Proves_i$ where i ranges over ordinals, but we omit the subscripts here.

Although there are many related works, we briefly compare our work with two of them, which, we think, are most closely connected.

Comparison with Allen's Work

Allen[16,17] interpreted Martin-Löf's type theory with universes in a type-free framework. He inductively defined a binary predicate τ which takes a term T and a binary predicate ϕ. Intuitively, $\tau(T, \phi)$ means "T is a type, and ϕ is an equivalence relation on the type T" (therefore a partial equivalence relation on the whole domain). For some fixed τ, the membership relation $a \in A$ in Martin-Löf's type theory will be interpreted by $\tau(\dot{A}, \phi) \wedge \phi(\dot{a}, \dot{a})$ for some ϕ where A and a are interpreted by $\dot{A}$ and $\dot{a}$. Having this interpretation in mind, τ is inductively defined. This inductive definition does not contain any negative occurrence of predicate variables being defined, hence is monotone in the usual sense. Allen constructed a model of Martin-Löf's type theory using this inductive definition. Since ϕ is a predicate over individuals (terms), τ is a second-order predicate, hence the whole theory becomes impredicative. Smith[18] formalized Allen's interpretation in **CTTR**, Martin-Löf's type theory with recursive types.

Allen's technique is superior to our theory in that he did not need any extra condition other than monotonicity while we need **HMONO-Q** condition. As a result, his theory might be more elegant if the only purpose is the interpretation of Martin-Löf's type theory. However, we believe that, our predicative theory has its own right because of the following reasons.

(1) In order to formalize something, we always look for as weak a theory as possible. Since a first-order theory with a least fixpoint operator (mu-calculus) is strictly weaker than the second order calculus[23], our first-order theory with half-monotone inductive definitions is strictly weaker than Allen and Smith's higher order framework. Our theory is therefore preferable in this respect.

(2) We believe that the form of our inductive definition is more natural and easier to understand than theirs.

In our theory, $\mathtt{Proves}(a, A)$ is directly defined by a half-monotone inductive definition, while in Allen's interpretation, $a \in A$ is defined as "$\tau(A, \phi) \wedge \phi(a, a)$ for some ϕ" where τ is defined by a monotone inductive definition (for a higher order predicate).

Comparison with Dybjer's Work

Dybjer[19,20] had a motivation which is similar to ours. He extended his recursive-induction mechanism[24] so that the universe hierarchy becomes definable in Martin-Löf-style type theory. As he mentioned, it can also incorporate the definition of $\langle \mathtt{Prop}, \mathtt{Proves} \rangle$. In some respects, his method is more flexible than ours, since he allows arbitrary recursive function for describing the second predicate (x in $\mathtt{Proves}(x, y)$) while we stick to make all definitions inductive, and also he required no extra conditions like **HMONO-Q** (syntactic conditions are necessary). His method was possible because, in Martin-Löf-style type theories, a clause in each Introduction Rule always introduces a constructor symbol, so that the domain could never be overlapped through the process of inductive generation. We are working in a type-free theory, and the inductive definition mechanism is applicable to a wider

In order to have the same kind of logic as internal and external ones, we may adopt $\mathbf{LTC}_0$ as our meta-theory, namely we may consider something like $\mathbf{LTC}_0 + \mathbf{HMID}$. It seems quite straightforward to move the **HMID** mechanism onto $\mathbf{LTC}_0$ The resulting theory $\mathbf{LTC}_0 + \mathbf{HMID}_0$ would subsume $\mathbf{LTC}_i$ $(i \geq 0)$ since we can internally define each P_i and T by half-monotone inductive definitions.

This approach (indicated by P. Dybjer) seems quite interesting. In fact, we plan to move our results to M. Sato's Reflective Proof Theory, which is closely related to **LTC**. This work in detail is for future work.

6.4. Interpretation of Martin-Löf's type theory

We can also interpret Martin-Löf's type theory **ITT** in our theory.

Since **ITT** can be interpreted by **LTC**[15], we can indirectly interpret **ITT** in our theory by using **LTC** as an intermediate theory. However, we can give a direct interpretation of **ITT** in our theory. The technique is just the same as in the previous section for **LTC**; we first define how all the terms in **ITT** is translated. We then define a predicate which means the set of (translated) terms by induction. Next, we define predicates which mean "A is a type", and "a is a member of a type A" simultaneously. Here, we have negative occurrences of the predicate variable corresponding to 'a is a member of a type A" when we interpret Σ and Π types, but it is treated just the same as in the definition of A_1 and B_1 for **LTC**. Finally, we can interpret a first universe in the same way as A_2 and B_2 for **LTC**. We can repeat this construction at finitely many times to interpret $\mathbf{ITT}_i$ for each $i \geq 0$.

We also have this interpretation is sound; namely if a judgement J is provable in $\mathbf{ITT}_i$, we can prove its translation in $\mathbf{BT{+}HMID}_0$.

Here, we omit the details for the lack of space.

7. Conclusion

We have studied a general form of half-monotone inductive definition in the context of a type-free first-order theory. It is an extension of monotone, simultaneous inductive definition, and is indispensable for *defining* provability relation within a theory, rather than a built-in relation. A characteristic point of this form is to allow a negative occurrence of a predicate variable which is being defined.

We changed the order between (tuples of) predicates so that such a kind of inductive definition is still monotone. The order is a slight modification of one proposed by Aczel[14], Hayashi and Nakano[4]. What is new in this paper is to give a general theory of such styles of inductive definitions, and also give a sound realizability interpretation to show our theory is really constructive. We have shown that the Logical Theory of Constructions[15] can be interpreted in our theory quite naturally. We have also shown that provability relations which are useful for program improvement are definable in our theory. Our formulation of $\mathbf{HMID}_0$ is sufficient for this purpose. As an application, we can prove equivalence of two definitions of provability relations, and can extract optimization programs by virtue of the program extraction theorem.

$$
\begin{aligned}
&\vee\ \exists y.\exists z.\ x = (y \dot{\wedge} z) \wedge Y(y) \wedge Y(z)\\
&\vee\ \exists y.\exists z.\ x = (y \dot{\vee} z) \wedge Y(y) \vee Y(z)\\
&\vee\ \exists y.\exists z.\ x = (y \dot{\rightarrow} z) \wedge X(y) \wedge (Y(y) \rightarrow Y(z))\\
&\vee\ \exists y.\ x = (\dot{\forall} y) \wedge \forall a.(\mathtt{Term}(a) \rightarrow Y(y(a)))\\
&\vee\ \exists y.\ x = (\dot{\exists} y) \wedge \exists a.(\mathtt{Term}(a) \wedge Y(y(a)))\\
&\vee\ \exists y.\ x = (\dot{\forall}^1 y) \wedge \forall f.(\mathtt{Func}(f) \rightarrow Y(y(f)))\\
&\vee\ \exists y.\ x = (\dot{\exists}^1 y) \wedge \exists f.(\mathtt{Func}(f) \wedge Y(y(f)))
\end{aligned}
$$

We take the least fixpoint as $\mu X.\langle A_0, B_0; X, Y\rangle$ and $\mu Y.\langle A_0, B_0; X, Y\rangle$, and call them $\mathtt{Prop}_0$ and $\mathtt{True}_0$. We omit the verification of **HMONO** and **HMONO-Q** conditions since it is almost the same as before.

Theorem 5 (Soundness of Translation) *Let F be a formula in* $\mathbf{LTC}_0$. *Then,* $\mathtt{Prop}_0(F')$ *is provable in* $\mathbf{BT{+}HMID}_0$, *and if F is provable in* $\mathbf{LTC}_0$, *then* $\mathtt{True}_0(F')$ *is provable in* $\mathbf{BT{+}HMID}_0$ *where F' is the translation of F.*

Each rule in $\mathbf{LTC}_0$ is easily justified. Since $\mathbf{LTC}_0$ does not have any induction rules, the restriction of the μ-ind-1 rule is not a problem.

We then go to the level 1. We assumed that the language of $\mathbf{LTC}_0$ already contained terms like $\dot{P}_i(a)$, so the languages of $\mathbf{LTC}_i$ are the same, and we continue to use the same definition of $\mathtt{Term}$ and so on.

We define the templates A_1 and B_1 as follows:

$$
\begin{aligned}
A_1 \ \triangleq\ & \lambda x.\ A_0(x)\\
&\quad \vee\ \exists a.\ \mathtt{Term}(a)\ \wedge\ x = \dot{P}_1(a)\\
&\quad \vee\ \exists a.\ \mathtt{Term}(a)\ \wedge\ x = \dot{T}(a)\\
B_1 \ \triangleq\ & \lambda x.\ B_0(x)\\
&\quad \vee\ \exists a.\ \mathtt{Term}(a)\ \wedge\ x = \dot{P}_1(a)\ \wedge\ \mathtt{Prop}_0(a)\\
&\quad \vee\ \exists a.\ \mathtt{Term}(a)\ \wedge\ x = \dot{T}(a)\ \wedge\ \mathtt{True}_0(a)
\end{aligned}
$$

We define $\mathtt{Prop}_1 \triangleq \mu X.\langle A_1, B_1; X, Y\rangle$ and $\mathtt{True}_0 \triangleq \mu Y.\langle A_1, B_1; X, Y\rangle$. Again, we omit the verification of **HMONO** and **HMONO-Q** conditions.

By repeating this kind of construction, we have the translation of $\mathbf{LTC}_i$ for each $i \geq 0$, and $\mathbf{LTC}_\omega$.

Theorem 6 (Soundness of Translation) *Let F be a formula in* $\mathbf{LTC}_i$ *for $i \geq 0$. Then,* $\mathtt{Prop}_i(F')$ *is provable in* $\mathbf{BT{+}HMID}_0$, *and if F is provable in* $\mathbf{LTC}_i$, *then* $\mathtt{True}_i(F')$ *is provable in* $\mathbf{BT{+}HMID}_0$ *where F' is the i-th level translation of F.*

6.3. Reformulating $\mathbf{HMID}_0$ *on* $\mathbf{LTC}_0$

Reflection in logic is internalization of a metatheory of the logic itself. In this respect, our theory is not really reflective, since our metatheory is $\mathbf{BT{+}HMID}_0$, while our target logic is **LTC**-like logic (where an implicational formula $a \rightarrow b$ may sometimes be a formula even if b is not a formula) up to now.

c	Φ_c^i	Ψ_c
$\dot{P}_j(a)$ $(with j < i)$	T	$P_j(a)$

$\mathbf{LTC}_i$ (for $i \geq 0$) is the theory defined as above. We therefore have an increasing hierarchy of theories $\mathbf{LTC}_0, \mathbf{LTC}_1, \cdots$. As a limit, we have $\mathbf{LTC}_\omega$, the union of $\mathbf{LTC}_i$ for $i \geq 0$. Note that we have no induction rules.

6.2. Interpreting LTC *by* BT+HMID$_0$

After defining the provability relation in the previous section, it is now straightforward to interpret $\mathbf{LTC}_0$ by $\mathbf{BT{+}HMID}_0$.

Variables in $\mathbf{LTC}_0$ is uniquely translated into variables in $\mathbf{BT{+}HMID}_0$. All the terms in $\mathbf{LTC}_0$ is injectively interpreted as some terms in $\mathbf{BT{+}HMID}_0$. For the sake of simplicity, we assume that terms like $\dot{P}_i(a)$ and $\dot{T}(a)$ are already contained in the language of $\mathbf{LTC}_0$ (with no rules about reflection).

To interpret the notion of terms, two predicates `Term` and `Func` are inductively defined; $\texttt{Term}(a)$ means a is an interpretation of a term, and $\texttt{Func}(a)$ means a is an interpretation of a function. Two more predicates are inductive defined; `Eval` and `FullEval`. They are used to interpret $\dot{\rightsquigarrow}$ and $\dot{\Rightarrow}$. We do not give their detailed definitions here.

Every formula in $\mathbf{LTC}_0$ is translated into a term in $\mathbf{BT{+}HMID}_0$ preserving the structure and the set of free variables. For example, $\exists y.\ x = Succ(y)$ is translated into $\dot{\exists}(\lambda y.\ (x \doteq SUCC(y)))$ where $Succ(_)$ is translated into $SUCC(_)$.

We now define two templates.

$$
\begin{aligned}
A_0 \triangleq \lambda x.\quad & x = \dot{\bot} \\
& \vee\ \exists a.\exists b.\ x = (a \doteq b) \\
& \vee\ \exists a.\exists b.\ x = (a \dot{\rightsquigarrow} b) \\
& \vee\ \exists a.\exists b.\ x = (a \dot{\Rightarrow} b) \\
& \vee\ \exists y.\exists z.\ x = (y \dot{\wedge} z) \wedge X(y) \wedge X(z) \\
& \vee\ \exists y.\exists z.\ x = (y \dot{\vee} z) \wedge X(y) \wedge X(z) \\
& \vee\ \exists y.\exists z.\ x = (y \dot{\rightarrow} z) \wedge X(y) \wedge (Y(y) \rightarrow X(z)) \\
& \vee\ \exists y.\ x = (\dot{\forall} y) \wedge \forall a.(\texttt{Term}(a) \rightarrow X(y(a))) \\
& \vee\ \exists y.\ x = (\dot{\exists} y) \wedge \forall a.(\texttt{Term}(a) \rightarrow X(y(a))) \\
& \vee\ \exists y.\ x = (\dot{\forall}^1 y) \wedge \forall f.(\texttt{Func}(f) \rightarrow X(y(f))) \\
& \vee\ \exists y.\ x = (\dot{\exists}^1 y) \wedge \forall f.(\texttt{Func}(f) \rightarrow X(y(f)))
\end{aligned}
$$

$$
\begin{aligned}
B_0 \triangleq \lambda x.\quad & X(x) \wedge \\
& (\exists a.\exists b.\ x = (a \doteq b) \wedge (a = b) \\
& \vee\ \exists a.\exists b.\ x = (a \dot{\rightsquigarrow} b) \wedge \texttt{Eval}(a,b) \\
& \vee\ \exists a.\exists b.\ x = (a \dot{\Rightarrow} b) \wedge \texttt{FullEval}(a,b)
\end{aligned}
$$

Terms of **LTC** is essentially combinatory terms with primitive function symbols for natural numbers, lambda terms, and others. Some of them such as $0, Succ, \lambda, \langle _, _ \rangle$, Inl, Inr are canonical. There are non-canonical function symbols corresponding to these canonical symbols. There are constants which internally represent logical connectives such as $\dot{\wedge}, \dot{\rightarrow}$ and so on. Atomic formulas are $\bot$, $a = b$, $a \rightsquigarrow b$, $a \Rightarrow b$. For a typographic reason, we use a different symbol $\Rightarrow$ from the original paper[15]. In $\mathbf{LTC}_i$ for $i \geq 1$, we have two more atomic formulas $T(a)$ and $P_j(a)$ (for $j \leq i$).

We briefly summarize intuitive meaning of atomic formulas. The formula $a = b$ means a and b are *definitionally* equal. The evaluation mechanism is call-by-name, and it is captured by $\rightsquigarrow$. The formula $a \rightsquigarrow b$ means that a term a evaluates to a canonical form b. Since this relation $\rightsquigarrow$ evaluates a term into a canonical form only (a canonical term is a term whose outermost function symbol, and it possibly contains redexes), we have another relation $\Rightarrow$ which represents "full evaluation". Two atomic formulas are important for reflection: "a term a is a proposition" (denoted as $P_i(a)$) and "a term a is a true proposition"(denoted as $T(a)$). Since reflection is repeated at finitely many times, $P_i(a)$ has a suffix i which represents the level of reflection. Besides them, there are ordinary first order logical connectives as well as quantifiers for functions $\forall^1$ and $\exists^1$. The predicate for natural numbers $Nat(a)$ can be defined as $\exists n.(a \Rightarrow n)$.

The logic of $\mathbf{LTC}_0$ is an ordinary intuitionistic predicate calculus with equality.

Extending $\mathbf{LTC}_0$ by reflection, we get $\mathbf{LTC}_1$. The rules (called reflection rules) for $P_1(a)$ and $T(a)$ are as follows. These rules give connection of internal logic (a in $T(a)$) to external logic (logic of $\mathbf{LTC}_0$).

$$\frac{\Phi_c^1}{P_1(c)} \qquad \frac{\Phi_c^1}{T(c) \leftrightarrow \Psi_c}$$

where Φ_c^1 and Ψ_c are defined by the following table.

c	Φ_c^1	Ψ_c
$(a \dot{=} b)$	T	$(a = b)$
$(a \dot{\rightsquigarrow} b)$	T	$(a \rightsquigarrow b)$
$(a \dot{\Rightarrow} b)$	T	$(a \Rightarrow b)$
$\dot{\bot}$	T	$\bot$
$(a \dot{\wedge} b)$	$P_1(a) \wedge P_1(b)$	$T(a) \wedge T(b)$
$(a \dot{\vee} b)$	$P_1(a) \wedge P_1(b)$	$T(a) \vee T(b)$
$(a \dot{\rightarrow} b)$	$P_1(a) \wedge (T(a) \rightarrow P_1(b))$	$T(a) \rightarrow T(b)$
$(\dot{\forall}(p))$	$\forall x.P_1(p(x))$	$\forall x.T(p(x))$
$(\dot{\exists}(p))$	$\forall x.P_1(p(x))$	$\exists x.T(p(x))$
$(\dot{\forall}^1(p))$	$\forall^1 f.P_1(p(f))$	$\forall^1 f.T(p(f))$
$(\dot{\exists}^1(p))$	$\forall^1 f.P_1(p(f))$	$\exists^1 f.T(p(f))$

Next, we give rules for $\mathbf{LTC}_i$ for $i > 1$. In the following, Φ_c^i and Ψ_c^i is in a row of the above table or the following new table.

$$\frac{\Phi_c^i}{P_i(c)} \qquad \frac{\Phi_c^i}{T(c) \leftrightarrow \Psi_c} \qquad \frac{P_j(e)}{P_i(e)}\ (j < i)$$

Suppose A and B are Harrop propositions. This formula contains disjunction and the existential quantifier, so it is not Harrop by definition. However, its proof carries no more information than x. We can, therefore, add this kind of propositions to Harrop propositions, and successfully reduce the program (proof).

Moreover, we can proceed to optimization of arbitrary self realizing formulas[c]. If we know a formula is self-realizing (which is a *semantic* notion, on the contrary to Harrop formula, which is a *syntactic* notion in ordinary logic.), then a realizer for this formula is redundant and we can eliminate it. This kind of optimization seems to have practical use, yet we have not studied this topic in detail.

In constructive programming, we often encounter inefficient programs. As is shown in the previous subsections, we can define several kinds of $\langle \mathtt{Prop}, \mathtt{Proves} \rangle$ in the theory $\mathbf{BT{+}HMID}_0$, and can prove equivalence of them. This corresponds to re-defining a new realizability interpretation, and proving its soundness, so the results of optimization may be the same. However, the point here is that we can prove meta-theorems such as equivalence of two definitions *within* the theory. Note that, we can further extract an optimization program from the meta-theorems by the program extraction theorem, which follows from the realizability interpretation. This point is one of major good points in our theory. For example:

Corollary 3 *In the previous theorem, we can effectively find functions f and g from the proof.*

This is an immediate consequence of the previous theorem together with the program extraction theorem.

Extracting programs from proofs as well as meta-proofs seems a quite promising paradigm in constructive programming. In this paper, we just presented foundation of a candidate theory, realizability interpretation, and its relation to other theories. However, we may be able to do *reflective constructive programming* in future based on the technique presented in this paper.

6. Interpretation of Other Theories

In this section, we show how other theories can be interpreted by the $\mathbf{HMID}_0$ mechanism. We first interpret the Logical Theory of Constructions, and then we briefly mention how to interpret Martin-Löf's type theory.

6.1. The Logical Theory of Constructions

The Logical Theory of Constructions (**LTC**)[15] is a special logic which has Aczel's Frege structures as its origin, and is used to interpret Martin-Löf's Type Theory **ITT**. We may say **LTC** is a formalized version of (a generalization of) Frege structures.

LTC is actually a family of theories, $\mathbf{LTC}_0, \mathbf{LTC}_1, \cdots, \mathbf{LTC}_\omega$. $\mathbf{LTC}_0$ is the basic theory, and $\mathbf{LTC}_{i+1}$ is a reflected version of $\mathbf{LTC}_i$. $\mathbf{LTC}_\omega$ is the union of each finite level of reflection.

[c] A self realizing formula is a formula $A(x)$, for which there exists a term f, the formula $\forall x.\ f(x)\ \mathbf{q}\ A$ holds.

$$\vee\ \exists y.\exists z.\ x = (y \dot{\wedge} z) \wedge X(y) \wedge X(z)$$
$$\vee\ \exists y.\exists z.\ x = (y \dot{\rightarrow} z) \wedge \mathtt{Prop}(y) \wedge X(z)$$
$$\vee\ \exists y.\ x = (\dot{\forall} y)) \wedge \forall a.X(y(a))$$

This is a usual positive inductive definition, so **HMONO**$(H; X)$ and **HMONO-Q**$(H; X)$ are clearly satisfied. Let $\mathtt{Harrop}$ be $\mu X.H$.

We can prove the following important properties of Harrop formulas.

Theorem 3

1. $\mathtt{Harrop}(x) \rightarrow \mathtt{Prop}(x)$

2. $\exists f.\forall x.(\mathtt{Harrop}(x) \rightarrow \mathtt{True}(x) \rightarrow \mathtt{Proves}(f, x))$

These are proved by the induction on $\mathtt{Harrop}$.

We then define the following new template B_1.

$$B_1 \ \stackrel{\triangle}{=}\ \lambda rx.\ B(x, r)\ \vee\ \mathtt{Harrop}(x) \wedge \mathtt{True}(x) \wedge r = \mathtt{nil}$$

Using $\langle A, B_1 \rangle$, we define a new provability relation.

Definition 12

$$\begin{aligned} \mathtt{Prop}_1 &\stackrel{\triangle}{=} \mu X.\langle A, B_1; X, Y\rangle \\ \mathtt{Proves}_1 &\stackrel{\triangle}{=} \mu Y.\langle A, B_1; X, Y\rangle \\ \mathtt{True}_1 &\stackrel{\triangle}{=} \lambda x.\exists r.\mathtt{Proves}_1(r, x) \end{aligned}$$

The pair $\langle A, B_1 \rangle$ also satisfies **HMONO** and **HMONO-Q**, so we can make use of μ-eq-1 and μ-ind-1.

We have the following theorem, which shows equivalence of two provability relations.

Theorem 4 *We have that*

$$\forall x.\ \mathtt{Prop}(x) \leftrightarrow \mathtt{Prop}_1(x),$$

and

$$\begin{aligned} \forall x.\ (\mathtt{Prop}(x) \rightarrow \exists f.\forall r.\ (\mathtt{Proves}(r, x) \rightarrow \mathtt{Proves}_1(f(r), x)) \\ \wedge\ \exists g.\forall r.\ (\mathtt{Proves}_1(r, x) \rightarrow \mathtt{Proves}(g(r), x))) \end{aligned}$$

Proof. (Sketch) Let $\mathtt{IH}(x)$ (standing for induction hypothesis) be the following formula:

$$\begin{aligned} &\mathtt{Prop}(x) \wedge \mathtt{Prop}_1(x) \\ &\wedge\ \exists f.\forall r.(\mathtt{Proves}(r, x) \rightarrow \mathtt{Proves}_1(f(r), x)) \\ &\wedge\ \exists g.\forall r.(\mathtt{Proves}_1(r, x) \rightarrow \mathtt{Proves}(g(r), x)) \end{aligned}$$

We can prove $\forall x.\mathtt{Prop}(x) \rightarrow \mathtt{IH}(x)$ and $\forall x.\mathtt{Prop}_1(x) \rightarrow \mathtt{IH}(x)$ by the induction on $\mathtt{Prop}$ and $\mathtt{Prop}_1$. Using Theorem 3, we can always create a candidate proof for a Harrop formula, hence we can calculate g from the proof of $\mathtt{Harrop}(x)$. □

We can improve the code further; for instance, we can optimize a program corresponding to a disjunctive formula if it is decidable. For example, let us see the following formula.

$$\lambda x.\ (x = 0 \wedge A(x)) \vee \exists yz.(x = \langle y, z\rangle \wedge B(x))$$

Here, $\mathtt{Nat}$ is the predicate defined before. Quantified formulas such as $\forall x.p(x)$ and $\exists x.p(x)$ are represented by $\dot{\forall}(\lambda x.p)$ and $\dot{\exists}(\lambda x.p)$, respectively. We can add other kinds of propositions such as $(a\downarrow)$. We did not do so simply because we want to show the essential feature of the mechanism.

Both templates contain negative occurrences of Y. **BT+HMID**$_0$ allows such occurrences provided **HMONO**$(A,B;X,Y)$ and **HMONO-Q**$(A,B;X,Y)$ are satisfied. Note that, in the above formulation, we bravely changed the order of the overlapping argument x and the non-overlapping argument r in Y. Strictly speaking, x must come first, but it can be adjusted easily.

Lemma 5 *For the above A and B,* **HMONO**$(A,B;X,Y)$ **HMONO-Q**$(A,B;X,Y)$ *hold.*

This lemma is proved in just the same as the case for Prime Numbers. The only negative occurrence of Y in A is in the subformula of the form $X(y)\wedge\forall w.(Y(w,y)\rightarrow\cdots)$, so $A_X[X_0,Y_0]$ is equivalent to $A_X[X_0,Y_1]$ provided that $\langle X_0,Y_0\rangle\leq_R\langle X_1,Y_1\rangle$. From this observation, we have $A_X[X_0,Y_0]\leq_N A_X[X_1,Y_1]$. Similarly, we have other cases and reach the conclusion.

From this lemma, we can safely define $\langle\mathtt{Prop},\mathtt{Proves}\rangle$ by the **HMID**$_0$ mechanism.

Definition 11

$$\begin{aligned}\mathtt{Prop} &\triangleq \mu X.\ \langle A,B;X,Y\rangle\\ \mathtt{Proves} &\triangleq \mu Y.\ \langle A,B;X,Y\rangle\\ \mathtt{True} &\triangleq \lambda x.\exists r.\ \mathtt{Proves}(r,x)\end{aligned}$$

The pair $\langle\mathtt{Prop},\mathtt{Proves}\rangle$ is a provability relation.

We can infer that $\mathtt{Prop}$ is the least fixpoint of A, while we cannot infer $\mathtt{Proves}$ is that of B, by the restriction for the μ-ind-1 rule. If we would omit the first conjunct $X(x)$ in the definition of B, the following (intuitively true) fact would not be provable in **BT+HMID**$_0$.

$$\forall x.(\mathtt{True}(x)\rightarrow\mathtt{Prop}(x))$$

However, we need induction only for $\mathtt{Prop}$ in practice, so μ-ind-1 is sufficient for all our needs.

5.2. Re-defining the Provability Relation

It is well known that Harrop formulas do not carry computational meaning. Hence, we may attach as a proof-term a dummy constant $\mathtt{nil}$ to Harrop formulas. This optimization technique is quite useful as is shown in (Ref. 4). Here we define a refined relation and prove equivalence of the original one and this version.

$$\begin{aligned}H \triangleq \lambda x.\quad & x=\dot{\bot}\\ & \vee\ \exists a.\exists b.\ x=(a\dot{=}b)\end{aligned}$$

5. Provability Relations and its Refinement

In this section, we define a basic version of provability relation and its variants using $\mathbf{HMID}_0$ mechanism.

5.1. Defining a Provability Relation

We shall simultaneously define a unary predicate `Prop` and a binary predicate `Proves` which formalizes the following two concepts.

`Prop`(p) represents "p is a proposition"
`Proves`(a,p) represents "a is a proof of p"

In ordinary predicate logic, the first concept is defined by itself, and does not depend on the second one. However, we are mainly interested in systems like Frege structures, Martin-Löf's Type Theory, or Sato's $\mathcal{RPT}$(Reflective Proof Theory[22]). One of the characteristic points of these theories is that the above two concepts essentially refers to each other, so we cannot define the first concept independently.

We will henceforth assume that all logical connectives are encoded by appropriate terms using corresponding constants, for instance, $a\dot{\wedge}b$ is $\langle \mathbf{c}_\wedge\, a\, b\rangle$ where $\mathbf{c}_\wedge$ is a constant uniquely associated with $\wedge$.

$$
\begin{aligned}
A \triangleq \lambda x.\quad & x = \dot{\bot} \\
& \vee\ \exists a.\exists b.\ x = a \doteq b \\
& \vee\ \exists a.\ x = \dot{N}(a) \\
& \vee\ \exists y.\exists z.\ x = (y \dot{\wedge} z) \wedge X(y) \wedge X(z) \\
& \vee\ \exists y.\exists z.\ x = (y \dot{\vee} z) \wedge X(y) \wedge X(z) \\
& \vee\ \exists y.\exists z.\ x = (y \dot{\rightarrow} z) \wedge X(y) \wedge \forall w.(Y(w,y) \rightarrow X(z)) \\
& \vee\ \exists y.\ x = (\dot{\forall} y) \wedge \forall a.X(y(a)) \\
& \vee\ \exists y.\ x = (\dot{\exists} y) \wedge \forall a.X(y(a))
\end{aligned}
$$

$$
\begin{aligned}
B \triangleq \lambda r x.\quad & X(x) \wedge \\
& (\exists a.\exists b.\ x = (a \doteq b) \wedge a = b \wedge r = \mathtt{nil} \\
& \vee\ \exists a.\ x = \dot{N}(a) \wedge \mathtt{Nat}(a) \wedge r = \mathtt{nil} \\
& \vee\ \exists y.\exists z.\exists a.\exists b.\ x = (y \dot{\wedge} z) \wedge r = \langle a,b\rangle \wedge Y(a,y) \wedge Y(b,z) \\
& \vee\ \exists y.\exists z.\exists a.\exists b.\ x = (y \dot{\vee} z) \wedge r = \langle a,b\rangle \wedge a\downarrow \\
& \qquad\qquad \wedge (a = \mathtt{true} \rightarrow Y(b,y)) \wedge (a \neq \mathtt{true} \rightarrow Y(b,z)) \\
& \vee\ \exists y.\exists z.\ x = (y \dot{\rightarrow} z) \wedge X(y) \wedge \mathtt{fun?}(r) = \mathtt{true} \\
& \qquad\qquad \wedge \forall w.(Y(w,y) \rightarrow Y(r(w),z)) \\
& \vee\ \exists y.\ x = \dot{\forall} y \wedge \forall a.\ Y(r(a), y(a)) \\
& \vee\ \exists y.\exists a.\exists b.\ x = \dot{\exists} y \wedge r = \langle a,b\rangle \wedge Y(a, y(b)))
\end{aligned}
$$

Let A and B be the same as in the prime number example.

Suppose $\langle X_0, X_0^*, Y_0, Y_0^* \rangle \leq_{R4} \langle X_1, X_1^*, Y_1, Y_1^* \rangle$. Then, our goal is to prove the following formula:

$$\langle A, \lambda nr.(r\ \mathbf{q}\ A(n)), B, \lambda ns.(s\ \mathbf{q}\ B(n)) \rangle \theta_0 \leq_{R4} \langle A, \lambda nr.(r\ \mathbf{q}\ A(n)), B, \lambda ns.(s\ \mathbf{q}\ B(n)) \rangle \theta_1$$

where θ_i is $\{X := X_i, X^* := X_i^*, Y := Y_i, Y^* := Y_i^*\}$.

Since X appears positively in A, it is easily seen that $A\theta_0 \leq_N A\theta_1$ and $\lambda nr.(r\ \mathbf{q}\ A(n))\theta_0 \leq_N \lambda nr.(r\ \mathbf{q}\ A(n))\theta_1$ hold. We also have $\langle A, B \rangle \theta_0 \leq_R \langle A, B \rangle \theta_1$, which is the **HMONO** condition.

The remaining goal is to prove the following:

$$\langle B, \lambda ns.(s\ \mathbf{q}\ B(n)) \rangle \theta_0 \leq_R \langle B, \lambda ns.(s\ \mathbf{q}\ B(n)) \rangle \theta_1$$

This is conjunction of three formulas, but here we shall prove the following one only.

$$\lambda ns.(s\ \mathbf{q}\ B(n))\theta_0 \leq_N \lambda ns.(s\ \mathbf{q}\ B(n))\theta_1$$

The formula $(s\ \mathbf{q}\ B(n))$ is expanded to the following:

$$\begin{array}{l} X(n) \wedge X^*(n, \mathtt{car}(s)) \wedge (\mathtt{cadr}(s)\ \mathbf{q}\ (1 < x)) \\ \quad \wedge \forall m, r.\ ((m < n) \wedge (r\ \mathbf{q}\ m < n) \rightarrow \\ \qquad X(m) \wedge X^*(m, \mathtt{car}(\mathtt{cddr}(s)mr)) \\ \qquad \wedge\ \forall u.(Y(m) \wedge Y^*(m,u)\ \rightarrow\ (\mathtt{cdr}(\mathtt{cddr}(s)mr)u\ \mathbf{q}\ \neg \mathtt{Div}(m,n)))). \end{array}$$

Since the only occurrence of Y^* appears in the subformula $Y(m) \wedge Y^*(m,u)$ and $Y_0(z)$ implies $\forall u. Y_0^*(z,u) \leftrightarrow Y_1^*(z,u)$, we have $\lambda ns.(s\ \mathbf{q}\ B(n))\theta_0 \leq_N \lambda ns.(s\ \mathbf{q}\ B(n))\theta_2$ where θ_2 is $\{X := X_0, X^* := X_0^*, Y := Y_0, Y^* := Y_1^*\}$. We also have $X_0(z)$ implies $Y_0(z) \leftrightarrow Y_1(z)$, so we have $\lambda ns.(s\ \mathbf{q}\ B(n))\theta_2 \leq_N \lambda ns.(s\ \mathbf{q}\ B(n))\theta_3$ where θ_3 is $\{X := X_0, X^* := X_0^*, Y := Y_1, Y^* := Y_1^*\}$. The occurrences of X and X^* are positive, and we finally have $\lambda ns.(s\ \mathbf{q}\ B(n))\theta_3 \leq_N \lambda ns.(s\ \mathbf{q}\ B(n))\theta_1$. Combining these results, we get $\lambda ns.(s\ \mathbf{q}\ B(n))\theta_0 \leq_N \lambda ns.(s\ \mathbf{q}\ B(n))\theta_1$, which is the conclusion.

Remark

This proof did not use the components of A and B in detail; it only mentioned the occurrences of X and Y. That the **HMONO**(A,B) and **HMONO-Q**(A,B) conditions hold come from the following facts:

- X appears positively in A and B
- Y may appear negatively in B, but the only negative occurrence of Y has a preceding occurrence of X.

The second fact can be made more precise: If every occurrence of Y is in a subformula of the form of $X(x) \wedge (Y(x,y) \rightarrow \cdots)$ where x is the overlapping argument, then we say Y has a preceding occurrence of X. (In the above example, we did not have an extra argument y in Y, but it is easily seen that the proof will not be affected by existence of such an argument.) We can see these two facts are keys to prove the two conditions, and whenever these two facts hold, we have the **HMONO** and **HMONO-Q** conditions.

One may think the verification of the **HMONO-Q** condition is too complex and hence our induction mechanism is hard to use. However, all the examples we have considered fall in this uniform pattern, and we believe its verification is not so problematic.

In the following, we will prove the goal (3). We fix $\mathbf{x}$ and r, and assume $(r\ \mathbf{q}\ A[X, M_Y])\theta$.

By expanding $m\ \mathbf{q}\ \mathbf{HMONO}(A, B)$, and putting

$$\begin{array}{rcl} X_0 & \triangleq & M_X \\ X_0^* & \triangleq & H \\ X_1 & \triangleq & F \\ X_1^* & \triangleq & \lambda \mathbf{x} r.(M_X(\mathbf{x}) \wedge L_X(\mathbf{x}, r) \wedge (r\ \mathbf{q}\ F(\mathbf{x}))) \\ Y_i & \triangleq & M_Y \quad (i = 0, 1) \\ Y_1^* & \triangleq & L_Y \quad (i = 0, 1) \end{array}$$

we get $M_X(\mathbf{x}) \wedge L_X(\mathbf{x}, r) \rightarrow (t\ \mathbf{q}\ A)\phi$ where

$$\phi \triangleq \{X := F, X^* := \lambda \mathbf{x} r.\ (M_X(\mathbf{x}) \wedge L_X(\mathbf{x}, r) \wedge (r\ \mathbf{q}\ F(\mathbf{x}))), Y := M_Y, Y^* := L_Y\}$$

and

$$t \triangleq (\mathtt{car}(m(\langle f, \langle \lambda \mathbf{x} \mathbf{y} s.s, \lambda \mathbf{x} \mathbf{y} r s.s \rangle \rangle)) \mathbf{x} r$$

By the simple monotonicity in X_0^*, we may replace ϕ by

$$\psi \triangleq \{X := F, X^* := \lambda \mathbf{x} r.(r\ \mathbf{q}\ F(\mathbf{x})), Y := M_Y, Y^* := L_Y\}$$

Note that $(t\ \mathbf{q}\ A)\psi$ is equivalent to $(t\ \mathbf{q}\ A[F, M_Y])$. From (1) and this formula, we have $M_X(\mathbf{x}) \wedge L_X(\mathbf{x}, r) \rightarrow \mathtt{car}(a)\mathbf{x}t\ \mathbf{q}\ F(\mathbf{x})$. We already have (2), so we can delete $L_X(\mathbf{x}, r)$ from this formula. By the fixpoint theorem, we can take f as a fixpoint of the equation $f\mathbf{x}r = \mathtt{car}(a)\mathbf{x}t$. Then we have $M(\mathbf{x}) \rightarrow (f\mathbf{x}r\ \mathbf{q}\ F(\mathbf{x}))$ which is the conclusion of the goal (3). □

By these lemmas, we have the soundness theorem. We also have the following corollaries.

Corollary 1 (Disjunction Property and Term-Existence Property) *1. Disjunction Property holds in* **BT+HMID**$_0$*; namely, from a proof of the formula* $A \vee B$*, we can decide which of* A *or* B *holds.*

2. Term Existence Property holds in **BT+HMID**$_0$*; namely, from a proof of the formula* $\exists x.\ A(x)$*, we can effectively find a term* t *such that* $A(t)$ *holds.*

Corollary 2 (Program Extraction Theorem) *From a proof of the formula* $\forall x. \exists y.\ A(x, y)$ *in* **BT+HMID**$_0$*, we can effectively find a term* f *and the proof of* $\forall x.\ A(x, f(x))$ *in* **BT+HMID**.

By Program Extraction Theorem, we can derive programs in our theory **BT+HMID**, in the style of constructive programming.

4.3. Verification of the **HMONO-Q** *condition*

In this subsection, we verify the **HMONO-Q** condition for the prime number example in Section 3.6.

Proof. By the induction on the abstracts A and B, we can prove that the same term is a realizer of both sides of the μ-eq-1 rule. □

Lemma 4 *The μ-ind-1 rule is realized.*

Proof. Suppose that the following conditions hold for A and B.

$$\begin{aligned} &\mathbf{HMONO}(A,B)\\ m\ \mathbf{q}\ &\mathbf{HMONO}(A,B)\\ &\mathbf{HMONO\text{-}Q}(A,B) \end{aligned}$$

We also assume the following for an n-ary abstract F.

$$\begin{aligned} &\langle A[F,M_Y],B[F,M_Y]\rangle \leq_R \langle F,M_Y\rangle\\ a\quad \mathbf{q}\quad &\langle A[F,M_Y],B[F,M_Y]\rangle \leq_R \langle F,M_Y\rangle \end{aligned}$$

By μ-`ind-1`, the first one implies $M_X \leq_N F$, which in turn implies $M_X \leq_N A[F,M_Y]$.

The second one implies

$$\forall \mathbf{x} r.(A[F,M_Y] \wedge (r\ \mathbf{q}\ A[F,M_Y]) \rightarrow (\mathtt{car}(a)\mathbf{x}r\ \mathbf{q}\ F(\mathbf{x}))) \tag{1}$$

Since we assume **HMONO-Q**, the quadruple $\langle \lambda\mathbf{x}.A, \lambda\mathbf{x}r.r\ \mathbf{q}\ A, \lambda\mathbf{x}\mathbf{y}.B, \lambda\mathbf{x}\mathbf{y}s.s\ \mathbf{q}\ B\rangle$ has a least fixpoint. We know that the first and the third elements of the least fixpoint are M_X and M_Y by Lemma 2. We abbreviate the second and the fourth elements as L_X and L_Y.

Our goal is to show the following formula:

$$\forall \mathbf{x} r.\ M_X(\mathbf{x}) \wedge L_X(\mathbf{x},r) \rightarrow (f\mathbf{x}r\ \mathbf{q}\ F(\mathbf{x}))$$

for an appropriate term f.

Let H be $\lambda\mathbf{x}r.(L_X(\mathbf{x},r) \wedge (M_X(\mathbf{x}) \rightarrow (f\mathbf{x}r\ \mathbf{q}\ F(\mathbf{x}))))$. Applying the quadruple $\langle M_X, H, M_Y, L_Y\rangle$ to the μ-ind-0 rule (in the case of R4-order) with a little calculation, it becomes:

$$\frac{\lambda\mathbf{x}r.(r\ \mathbf{q}\ A[X,M_Y])\theta \leq_N H}{L_X \leq_N H}$$

where θ is $\{X := M_X, X^* := H\}$. Then all we have to prove is

$$\forall \mathbf{x}, r.\ (r\ \mathbf{q}\ A[X,M_Y])\theta \rightarrow H(\mathbf{x},r)$$

for an appropriate term f. This is equivalent to conjunction of the following formulas:

$$\forall \mathbf{x}, r.\ (r\ \mathbf{q}\ A[X,M_Y])\theta \rightarrow L_X(\mathbf{x},r) \tag{2}$$

$$\forall \mathbf{x}, r.\ (r\ \mathbf{q}\ A[X,M_Y])\theta \rightarrow M_X(\mathbf{x}) \rightarrow (f\mathbf{x}r\ \mathbf{q}\ F(\mathbf{x})) \tag{3}$$

By **HMONO-Q** condition, we have the simple monotonicity for $(r\ \mathbf{q}\ A[X,Y])$ in X^*. Hence, we can replace H by L_X in θ, and we obtain the goal (2).

extended to the general case in the same pattern as these clauses, but we have not done the details since the theory **BT+HMID**$_0$ is sufficient for our needs.

If the formula $(e\ \mathbf{q}\ A)$ is proved, we say "the formula A is provably realized by the term e". Now we can define the condition **HMONO-Q**.

Definition 10 (HMONO-Q) *The condition* **HMONO-Q**$(\lambda\mathbf{x}.A, \lambda\mathbf{xy}.B)$ *is defined as the quadruple of abstracts* $\langle \lambda\mathbf{x}.A, \lambda\mathbf{x}r.(r\ \mathbf{q}\ A), \lambda\mathbf{xy}.B, \lambda\mathbf{xy}s.(s\ \mathbf{q}\ B)\rangle$ *is monotone wrt. R4-order.*

We write $\mu X.\langle \lambda\mathbf{x}.A, \lambda\mathbf{x},\mathbf{y}.B\rangle$ and $\mu Y.\langle \lambda\mathbf{x}.A, \lambda\mathbf{x},\mathbf{y}.B\rangle$ as L_0 and L_1, and $\mu X.Q$, $\mu X^*.Q$, $\mu Y.Q$, and $\mu Y^*.Q$ as M_0, M_1, M_2 and M_3 where Q is $\langle \lambda\mathbf{x}.A, \lambda\mathbf{x}r.(r\ \mathbf{q}\ A), \lambda\mathbf{xy}.B, \lambda\mathbf{xy}s.(s\ \mathbf{q}\ B)\rangle$

Lemma 2 *Assuming the* **HMONO** *and* **HMONO-Q** *conditions hold, we have that* $L_0(\mathbf{x}) \leftrightarrow M_0(\mathbf{x})$ *and* $L_1(\mathbf{x},\mathbf{y}) \leftrightarrow M_2(\mathbf{x},\mathbf{y})$

Proof. We easily have $\langle L_0, L_1\rangle \leq_R \langle M_0, M_2\rangle$. We shall prove the opposite direction. We have $\langle L_0, M_1, L_1, M_3\rangle \leq_{R4} \langle M_0, M_1, M_2, M_3\rangle$. Applying these elements to $\langle \lambda\mathbf{x}.A, \lambda\mathbf{x}r.(r\ \mathbf{q}\ A), \lambda\mathbf{xy}.B, \lambda\mathbf{xy}s.(s\ \mathbf{q}\ B)\rangle$, we have

$$\langle L_0, \lambda\mathbf{x}r.(r\ \mathbf{q}\ A)\theta, L_1, \lambda\mathbf{xy}s.(s\ \mathbf{q}\ B)\theta\rangle \leq_{R4} \langle M_0, M_1, M_2, M_3\rangle$$

where θ is $\{X := L_0, X^* := M_1, Y := L_1, Y^* := M_3\}$. By the definition of $\leq_{R4}$ and $\langle L_0, L_1\rangle \leq_R \langle M_0, M_2\rangle$, we can replace the right-hand-side by $\langle L_0, M_1, L_1, M_3\rangle$, and get

$$\langle L_0, \lambda\mathbf{x}r.(r\ \mathbf{q}\ A)\theta, L_1, \lambda\mathbf{xy}s.(s\ \mathbf{q}\ B)\theta\rangle \leq_{R4} \langle L_0, M_1, L_2, M_3\rangle$$

Taking this formula as an assumption of the μ-ind-1 rule (besides the **HMONO** and **HMONO-Q** conditions), we obtain $\langle M_0, M_1, M_2, M_3\rangle \leq_{R4} \langle L_0, M_1, L_1, M_3\rangle$ by μ-ind-1. This formula implies $\langle M_0, M_2\rangle \leq_R \langle L_0, L_1\rangle$. □

4.2. Soundness of Realizability Interpretation

We will prove the following soundness theorem.

Theorem 2 (Soundness of q-realizability for BT+HMID$_0$)
Let $A_1 \cdots, A_n, F$ *be formulas. If*

$$A_1 \cdots, A_n \vdash_{\mathbf{BT+HMID}_0} F$$

holds, then we have

$$A_1, \cdots, A_n, (x_1\ \mathbf{q}\ A_1), \cdots, (x_n\ \mathbf{q}\ A_n) \vdash_{\mathbf{BT+HMID}} r\ \mathbf{q}\ F$$

for some term r.

Since the only essential difference between Tatsuta's theory and our theory is **HMID**$_0$, we only have to consider **HMID**$_0$ to prove this theorem.

Lemma 3 *Suppose that the following conditions hold for* A *and* B.

$$\mathbf{HMONO}(A, B)$$
$$\mathbf{HMONO\text{-}Q}(A, B)$$

Then, the μ*-eq-1 rule is realized.*

4.1. Realizability Interpretation

We assume that, with each predicate variable X of arity n, a predicate variable X^* of arity $n+1$ is uniquely associated. X^* will be used for representing the abstract $\lambda \mathbf{x} e.\ e\ \mathbf{q}\ X(\mathbf{x})$.

Given a term e and a formula A in $\mathbf{BT{+}HMID}_0$, the $\mathbf{q}$-realizability $e\ \mathbf{q}\ A$ is a formula which is defined by induction on the structure of A as follows:

1. $e\ \mathbf{q}\ A \stackrel{\triangle}{=} e = \mathtt{nil} \wedge A$ where A is atomic.
2. $e\ \mathbf{q}\ Z(\mathbf{t}) \stackrel{\triangle}{=} Z(\mathbf{t}) \wedge Z^*(\mathbf{t}, e)$ where Z is a predicate variable.
3. $e\ \mathbf{q}\ A \wedge B \stackrel{\triangle}{=} (\mathtt{car}(e)\ \mathbf{q}\ A) \wedge (\mathtt{cdr}(e)\ \mathbf{q}\ B)$
4. $e\ \mathbf{q}\ A \vee B \stackrel{\triangle}{=} (\mathtt{car}(e)) \downarrow \wedge\ (\mathtt{car}(e) = \mathtt{true} \rightarrow (\mathtt{cdr}(e)\ \mathbf{q}\ A))$
   $\wedge\ (\mathtt{car}(e) \neq \mathtt{true} \rightarrow (\mathtt{cdr}(e)\ \mathbf{q}\ B))$
5. $e\ \mathbf{q}\ A \rightarrow B \stackrel{\triangle}{=} \forall r.\ A \wedge (r\ \mathbf{q}\ A) \rightarrow (e(r)\ \mathbf{q}\ B)$
6. $e\ \mathbf{q}\ \forall x.A \stackrel{\triangle}{=} \forall x.(e(x)\ \mathbf{q}\ A)$
7. $e\ \mathbf{q}\ \exists x.A \stackrel{\triangle}{=} (\mathtt{cdr}(e)\ \mathbf{q}\ A_x[\mathtt{car}(e)]) \wedge A_x[\mathtt{car}(e)]$
8. $e\ \mathbf{q}\ (\lambda x.A)(t) \stackrel{\triangle}{=} e\ \mathbf{q}\ A_x[t]$
9. $e\ \mathbf{q}\ (\mu X.\langle \lambda \mathbf{x}.A, \lambda \mathbf{xy}.B\rangle(\mathbf{t})) \stackrel{\triangle}{=}$
   $(\mu X^*.\langle \lambda \mathbf{x}.A, \lambda \mathbf{x}r.(r\ \mathbf{q}\ A), \lambda \mathbf{xy}.B, \lambda \mathbf{xy}s.(s\ \mathbf{q}\ B); X, X^*, Y, Y^*\rangle)(\mathbf{t}, e)$
10. $e\ \mathbf{q}\ (\mu Y.\langle \lambda \mathbf{x}.A, \lambda \mathbf{xy}.B\rangle(\mathbf{t}, \mathbf{u})) \stackrel{\triangle}{=}$
    $(\mu Y^*.\langle \lambda \mathbf{x}.A, \lambda \mathbf{x}r.(r\ \mathbf{q}\ A), \lambda \mathbf{xy}.B, \lambda \mathbf{xy}s.(s\ \mathbf{q}\ B); X, X^*, Y, Y^*\rangle)(\mathbf{t}, \mathbf{u}, e)$

All but clauses 2, 9, and 10 are quite standard. Let us explain these three definitions following Tatsuta.

Clause 2 is for a predicate variable which will be used through inductive definition. Clauses 9 and 10 represent the realizability of inductively defined predicates. The realizability interpretation for inductively defined predicates $\mu X.A$ is defined so that the realizer of $(\mu X.A)(t)$ and that of $A_X[\mu X.A](t)$ are the same. So we want to define $e\ \mathbf{q}\ (\mu X.A)(t)$ as $e\ \mathbf{q}\ A_X[\mu X.A](t)$. However, by expanding $e\ \mathbf{q}\ A_X[\mu X.A](t)$, we shall encounter $e\ \mathbf{q}\ (\mu X.A)(s)$, which we are defining now. By defining $X^*(t, e)$ as $e\ \mathbf{q}\ X(t)$, we have another inductive definition $X^*(t, e) \leftrightarrow (e\ \mathbf{q}\ A(t))$, which may contain X^* recursively. (This is a valid definition by the **HMONO-Q** condition, which ensures the monotonicity in this form.) This is the intuition of the clauses 9 and 10.

Remark

We defined the clauses 9 and 10 for the R-order only, and not for the general $R(n, df)$-order, since our realizability interpretation is for $\mathbf{BT{+}HMID}_0$, and not for the full theory **BT+HMID**. We think our realizability interpretation may be

At the next stage, we want to replace it by the following one:

$$\mathtt{Prime}(n) \leftrightarrow \mathtt{Nat}(n) \ \wedge \ 1 < n \ \wedge \ \forall m < n.(\mathtt{Prime}(m) \rightarrow \neg \ \mathtt{Div}(m, n))$$

Since the latter contains a negative occurrence of `Prime`, it is not directly acceptable by conventional theories.

On the contrary, we will show that we can define it by **HMID** as follows.

$$\begin{aligned} A &\stackrel{\triangle}{=} \lambda n.\ n = 0 \ \vee \ \exists m.\ n = \mathtt{suc}(m) \wedge X(m) \\ B &\stackrel{\triangle}{=} \lambda n.\ X(n) \ \wedge \ 1 < n \\ &\qquad \wedge \forall m < n.\ (X(m) \wedge (Y(m) \rightarrow \neg \ \mathtt{Div}(m, n))) \end{aligned}$$

In this example, the argument n is the overlapping argument for A and B. We assume here that $<$ and `Div` are defined in a standard way. The second one $B(n)$ is intended to mean that n is a prime number.

The defining formula for A is the same as one for `Nat`, so one may think it is unnecessary. However, this is the trick to use the **HMID** mechanism. In this definition, $A(n)$ determines a *domain*, and $B(n)$ determines a prime number in the domain. The domain gradually increases by repeated application of A. We have to use dependency on the argument n between two predicates in order to guarantee the monotonicity with respect to the refined order.

For these A and B, the **HMONO**(A, B) and **HMONO-Q**(A, B) conditions hold as shown below. Hence, we have the least fixpoint $\langle M_X, M_Y \rangle$ for the equation $\langle A(x), B(x) \rangle \leftrightarrow \langle X(x), Y(x) \rangle$. We define `Prime` as M_Y. Using the induction rule, $M_X(x) \leftrightarrow \mathtt{Nat}(x)$ is easily proved, and we get

$$\mathtt{Prime}(n) \ \leftrightarrow \ \mathtt{Nat}(n) \ \wedge \ 1 < n \ \wedge \ \forall m < n.(\mathtt{Prime}(m) \rightarrow \neg \ \mathtt{Div}(m, n))$$

as a desired result.

Verification of the HMONO(A, B) condition

Suppose that $\langle X_0, Y_0 \rangle \leq_R \langle X_1, Y_1 \rangle$ holds. Since the defining formula for A contains only a positive occurrence of X, it is easily seen $A_X[X_0] \leq_N A_X[X_1]$. From the assumption, we have $\forall z.\ (X_0(z) \rightarrow (Y_0(z) \leftrightarrow Y_1(z)))$. Hence $B_{X,Y}[X_0, Y_0]$ is equivalent to $B_{X,Y}[X_0, Y_1]$, and we have $B_{X,Y}[X_0, Y_0] \leq_N B_{X,Y}[X_1, Y_1]$. We also have, for an arbitrary m such that $X_0(m)$ holds, $B_{X,Y}[X_0, Y_0](m) \leftrightarrow B_{X,Y}[X_1, Y_1](m)$, and we have the conclusion.

The verification of the **HMONO-Q**(A, B) condition is postponed until Section 4.3.

4. Realizability interpretation and its soundness for BT+HMID$_0$

In this section, we define a realizability interpretation for **BT+HMID**$_0$ in **BT+HMID**, and prove its soundness. We use so called **q**-realizability interpretation[13] instead of **r**-realizability. The following interpretation is essentially the same as Tatsuta's one except that we use total term logic instead of partial term logic, and that the inductive definition is extended to the R-order.

As for the second clause: let Δ be a directed set in Dom. Let L_i be $\cup\{\mathbf{X}_i \mid \langle \mathbf{X}_1, \cdots, \mathbf{X}_n\rangle \in \Delta\}$. We will prove that $\langle L_1, \cdots, L_n\rangle$ is the lub of Δ.

Suppose it is not an upper bound of Δ. Then there exists $\langle S_1, \cdots, S_n\rangle \in \Delta$ such that $\langle S_1, \cdots, S_n\rangle \sqsubseteq \langle L_1, \cdots, L_n\rangle$ does not hold. From the definition of L_i, we have that $L_i|S_{df(i)}$ is properly larger than $S_i|S_{df(i)}$ for some i. Then we have that there exists $\langle T_1, \cdots, T_n\rangle \in \Delta$ such that $T_i|\{\mathbf{s}\}$ is properly larger than $S_i|\{\mathbf{s}\}$ for some i and some $\mathbf{s} \in S_{df(i)}$. But this contradicts to the directedness of Δ, since we do not have an upper bound of $\langle S_1, \cdots, S_n\rangle$ and $\langle T_1, \cdots, T_n\rangle$. The minimality is apparent from the definition.

We can also prove that $\langle \emptyset, \cdots, \emptyset\rangle$ is the least element in Dom. $\square$

In contrary to the set inclusion order, a least upper bound for an arbitrary subset wrt $\sqsubseteq_{R(n,df)}$ does not always exist, since the union of two sets may have too many elements. In particular, the whole domain $\langle D^{m_1}, \cdots, D^{m_n}\rangle$ is not always greater than an arbitrary element in Dom.

We can interpret a monotone operator in **BT+HMID** by a monotone operator in the model. From the lemma, we can interpret each inductively defined predicate as a least fixpoint of the corresponding operator. It is easy to see the rules μ-eq-0 and μ-ind-0 are satisfied. As a consequence, we have the following theorem.

Theorem 1 (Consistency)
1. We can construct a model of **BT+HMID** *from any model of* **BT**.
2. **BT+HMID** *(therefore,* **BT+HMID**$_0$*) is consistent.*

Our model construction is based on classical set theory to ensure the existence of a least fixpoint for each monotone operator. Hence it does not differ so much from Aczel's[14] and Hayashi and Nakano's[4]. The aim of this section is just to show the consistency of our theory in its general form. Our claim that our theory is constructive in nature is not due to this model construction, but due to the realizability interpretation defined in the next section.

3.6. Example of HMID

Natural Numbers

`Nat`, the predicate for natural numbers, is defined as follows. Let A be the following template:

$$\lambda x.\ x = 0 \vee \exists y.\ (y = \mathtt{suc}(x) \wedge X(y))$$

where $\mathtt{suc}(x)$ is encoded by $\langle 0, x\rangle$. Then, `Nat` can be defined as $\mu X.A$. Since A is positive in X, **HMONO** and **HMONO-Q** conditions are automatically satisfied. The usual induction principle follows from μ-ind-1.

Prime Numbers

The next example is a predicate for prime numbers.

At first thought, it is defined by:

$$\mathtt{Prime}(n) \ \leftrightarrow\ \mathtt{Nat}(n) \ \wedge\ 1 < n \ \wedge\ \forall m.\ (1 < m < n \rightarrow \neg\, \mathtt{Div}(m, n))$$

where $\mathtt{Div}(m, n)$ means "m divides n".

in the definition of $\leq_R$:

$$\forall x, y.\ P_0(x) \rightarrow Q_1(x, y) \rightarrow Q_0(x, y)$$

This contrasts to the corresponding situation in type theory by Dybjer[20]. In his theory, the condition for the inductive definition is given by a purely syntactic way. We shall compare his theory and ours in the conclusion.

3.5. A model of **HMID**

Let us explain how to construct a model of **BT+HMID**.

We begin with an arbitrary model of **BT**; there is a domain D, and all the terms are interpreted as elements of D. An n-ary predicate is interpreted as a subset of D^n. Let $\mathcal{P}(S)$ be the powerset of S. Given a degree n, a dependency function df and a sequence of arities $m_1, \cdots, m_n$, we define an order $\sqsubseteq_{n,df}$ on Dom as follows, where Dom is $\mathcal{P}(D^{m_1}) \times \cdots \times \mathcal{P}(D^{m_n})$.

Definition 8 (Order $\sqsubseteq_{n,df}$) *For $S_i \subseteq D^{m_i}$, and $T_i \subseteq D^{m_i}$ $(1 \leq i \leq n)$, we define $\langle S_1, \cdots, S_n\rangle \sqsubseteq_{n,df} \langle T_1, \cdots, T_n\rangle$ if, for each i with $1 \leq i \leq n$, the conjunction of the following conditions hold.*

1. $S_{df(i)} \subseteq T_{df(i)}$
2. $S_i \subseteq T_i$
3. $T_i|S_{df(i)} = S_i|S_{df(i)}$

where $T|S \stackrel{\triangle}{=} \{\langle \mathbf{s}, \mathbf{t}\rangle \mid \mathbf{s} \in S \wedge \langle \mathbf{s}, \mathbf{t}\rangle \in T\}$

The order $\sqsubseteq_{n,df}$ just corresponds to the $R(n, df)$-order. We may sometimes omit the subscript in $\sqsubseteq_{n,df}$.

Definition 9 (Monotone Operator) *A sequence of functions $F_i : Dom \rightarrow \mathcal{P}(D^{m_i})$ (for $1 \leq i \leq n$) is monotone if and only if, for all $S_i, T_i \subseteq D^{m_i}$,*

$\langle S_1, \cdots, S_n\rangle \sqsubseteq_{n,df} \langle T_1, \cdots, T_n\rangle$ *implies*

$$\langle F_1(S_1, \cdots, S_n), \cdots, F_n(S_1, \cdots, S_n)\rangle \sqsubseteq_{n,df} \langle F_1(T_1, \cdots, T_n), \cdots, F_n(T_1, \cdots, T_n)\rangle$$

Lemma 1 *The following holds:*

1. *$\sqsubseteq$ is an order on Dom.*
2. *The least element exists in Dom. Moreover, every directed subset of Dom has a least upper bound (lub) wrt $\sqsubseteq$.*
3. *There exists a least fixpoint for each monotone operator wrt $\sqsubseteq$.*

Proof. The first clause of the lemma is straightforward. The second clause means that Dom with $\sqsubseteq$ is a complete partial order (cpo), which implies the third clause (in classical set theory).

where F and G are arbitrary n-ary and $n+m$-ary predicates.

As usual, the first rule states $\langle M_X, M_Y \rangle$ is a solution of an equation, and the second rule states the pair $\langle M_X, M_Y \rangle$ is the least solution with respect to the order $\leq_R$.

Similarly, we can define the μ-eq-0 and μ-ind-0 rules with respect to an arbitrary order $\leq_{R(n,df)}$.

The union of μ-eq-0 and μ-ind-0 for an arbitrary order $\leq_{R(n,df)}$ is called **HMID**. Since the order $\leq_{R(n,df)}$ is an extension of $\leq_N$, **HMID** is an extension of **MID**.

3.4. Restricted version of half-monotone inductive definition

As in **MID**, we need additional conditions for giving a sound realizability interpretation. The first condition **HMONO-Q** is a corresponding condition to **MONO-Q** in $\mathbf{MID}_0$. The second one is that, we can use the μ-ind rule only when the abstract G is the least fixpoint M_Y, and the order is $\leq_R$ (of degree 2). Under these conditions, the rules become:

$$\frac{\mathbf{HMONO}(A,B) \quad \mathbf{HMONO\text{-}Q}(A,B)}{\mu\text{-}\mathtt{eq}} \ (\mu\text{-}\mathtt{eq\text{-}1})$$

$$\frac{\begin{array}{l}\mathbf{HMONO\text{-}Q}(A,B) \\ \mathbf{HMONO}(A,B) \qquad \langle A[F,M_Y], B[F,M_Y] \rangle \leq_R \langle F, M_Y \rangle\end{array}}{M_X \leq_N F} \ (\mu\text{-}\mathtt{ind\text{-}1})$$

In the latter rule, we write two conditions horizontally for a typographic reason.

The precise form of **HMONO-Q**(A,B) is described in Section 4; here we just remark that **HMONO-Q**(A,B) becomes **MONO-Q**(A) when B is identically true.

Under the conditions **HMONO** and **HMONO-Q**, the first rule states that $\langle M_X, M_Y \rangle$ is a solution, and the second one states that it is the least one. Since μ-ind-1 restricts the use of induction principle only for the first predicate (F in the above definition), we cannot infer the minimality of M_Y.

Let us consider that the first predicate M_X defines a domain, and the second predicate M_Y defines a property on that domain. Then we need an induction for M_X, but not for M_Y. We have no interest in the behavior of $M_Y(\mathbf{x}, \mathbf{y})$ where $M_X(\mathbf{x})$ does not hold, since such an $\mathbf{x}$ is outside of the domain. In Sections 5 and 6, we shall show the above formulation is sufficient for most purposes.

The μ-`ind-1` rule subsumes the following more convenient form:

$$\frac{\mathbf{HMONO}(A,B) \ \mathbf{HMONO\text{-}Q}(A,B) \ A[F,M_Y] \leq_N F \ F \leq_N M_X}{M_X \leq_N F}$$

The union of the μ-eq-1 and μ-ind-1 rules (for $\leq_R$ order) is called $\mathbf{HMID}_0$. Note that $\mathbf{HMID}_0$ is an extension of $\mathbf{MID}_0$.

Remark

Note that, for our purposes, a positive inductive definition is worthless since we cannot have a meaningful syntactic condition which subsumes the third condition

a notion of a dependency function. For a natural number n, a dependency function df is a finite function from $\{1,\cdots,n\}$ to the set of natural numbers, which satisfies $0 \leq df(i) < i$. We say n is the degree of df.

Definition 6 (Refined order in general case) *Let df be a dependency function of degree n. For two sequences of predicates $\langle P^1,\cdots,P^n\rangle$ and $\langle Q^1,\cdots,Q^n\rangle$ whose arities correspond, we define $\langle P^1,\cdots,P^n\rangle \leq_{R(n,df)} \langle Q^1,\cdots,Q^n\rangle$ as:*

$$\langle P^{df(i)}, P^i\rangle \leq_R \langle Q^{df(i)}, Q^i\rangle \text{ holds for each } i\ (1 \leq i \leq n)$$

where P^0 and Q^0 are $\lambda\mathbf{x}.\bot$.

If $df(i) = 0$, the condition becomes $P^i \leq_N Q^i$. Note that, the orders $\leq_N$ and $\leq_R$ can be written as $\leq_{R(1,df_N)}$ and $\leq_{R(2,df_R)}$ where $df_N(1) = 0, df_R(1) = 0, df_R(2) = 1$.

Example

Let df be a dependency function of degree 4 with $df(1) = df(2) = 0, df(3) = 1$ and $df(4) = 3$. Then the order $\langle P_0,Q_0,R_0,S_0\rangle \leq_{R(4,df)} \langle P_1,Q_1,R_1,S_1\rangle$ is

$$P_0 \leq_N P_1 \ \wedge\ Q_0 \leq_N Q_1 \ \wedge\ \langle P_0,R_0\rangle \leq_R \langle P_1,R_1\rangle \ \wedge\ \langle R_0,S_0\rangle \leq_R \langle R_1,S_1\rangle.$$

We call this order $R4$-order.

3.3. Half-monotone inductive definition

We first extend the language of our theory. Let A_i and X_i be an n_i-ary abstract and an n_i-ary predicate variable for $1 \leq i \leq n$. Then, $\mu X_i.\langle A_1,\cdots,A_n; X_1,\cdots,X_n\rangle$ is an n_i-ary abstract with $X_1,\cdots,X_n$ bound by this μ operator. This form will be used for representing the i-th component of a least fixpoint.

The definition of monotonicity does not change, but we present it here for completeness.

Definition 7 (Monotonicity wrt. $\leq_R$) *Let $\langle A,B\rangle$ be a pair of n-ary and $n+m$-ary abstracts, each of which possibly contains n-ary and $n+m$-ary predicate variables X and Y. Then we define*

$$\mathbf{HMONO}(A,B;X,Y) \triangleq \langle X_0,Y_0\rangle \leq_R \langle X_1,Y_1\rangle \rightarrow \langle A_{X,Y}[X_0,Y_0], B_{X,Y}[X_0,Y_0]\rangle \leq_R \langle A_{X,Y}[X_1,Y_1], B_{X,Y}[X_1,Y_1]\rangle$$

We often omit X and Y in $\mathbf{HMONO}(A,B;X,Y)$. We may similarly define the **HMONO** condition for the case of the generalized order $\leq_{R(n,df)}$.

Let us abbreviate as follows:

$$M_X \triangleq \mu X.\langle A,B;X,Y\rangle$$
$$M_Y \triangleq \mu Y.\langle A,B;X,Y\rangle$$
$$\mu - \mathtt{eq} \triangleq \forall\mathbf{x}.\ (M_X(\mathbf{x}) \leftrightarrow A[M_X,M_Y](\mathbf{x})) \wedge \forall\mathbf{xy}.\ (M_Y(\mathbf{x},\mathbf{y}) \leftrightarrow B[M_X,M_Y](\mathbf{x},\mathbf{y}))$$

Then the inference rules for a half-monotone inductive definition are as follows:

$$\frac{\mathbf{HMONO}(A,B)}{\mu\text{-}\mathtt{eq}}\ (\mu\text{-}\mathtt{eq}\text{-}0)$$

$$\frac{\mathbf{HMONO}(A,B) \quad \langle A[F,G],B[F,G]\rangle \leq_R \langle F,G\rangle}{\langle M_X,M_Y\rangle \leq_R \langle F,G\rangle}\ (\mu\text{-}\mathtt{ind}\text{-}0)$$

$\mathbf{MID}_0$ is an extension of positive inductive definitions (where all the occurrences of X in A must be in syntactically positive positions), since every positive A satisfies $\mathbf{MONO}(A)$ and $\mathbf{MONO\text{-}Q}(A)$. Moreover, the extension is proper as shown by Tatsuta's example, $\mu X.\, \bot \wedge (X \to X)$, which is not positive, but trivially satisfies **MONO** and **MONO-Q** conditions. However, he has shown no substantial example which demonstrates the difference between positive and monotone inductive definitions. Therefore, practical use of his results was not known. In the following, we will extend his **MID** ($\mathbf{MID}_0$) to obtain **HMID** ($\mathbf{HMID}_0$, resplectively). Our definition of the realizability interpretation is essentially due to Tatsuta's. What is new is that our theory with $\mathbf{HMID}_0$ has many uses in defining notions such as the provability relation.

Tatsuta also gave a realizability interpretation to full **MID** in a second-order theory. This interpretation is quite straightforward, since one can directly define the least solution of $\forall \mathbf{x}.\, A(\mathbf{x}) \leftrightarrow X(\mathbf{x})$ as $\forall X.\, (\forall \mathbf{y}. A(\mathbf{y}) \to X(\mathbf{y})) \to X(\mathbf{x})$ in a second-order theory. He also discussed the difference of these two interpretations. Here, we do not get into the long-standing debate between first-order theories plus induction principles (intuitionistic mu-calculus) versus second-order theories with the above encoding of the least fixpoint. We just quote Tatsuta's remark that a realizer of **MID** in a first-order theory has a loop structure (a recursive call) inside it, while one in a second-order theory (via the above encoding) does not have a loop structure. See (Ref. 21) for details.

3.2. Refined order

Here we define the refined order discussed at the beginning of this section, and will use it in inductive definitions.

Definition 5 (Refined order[b]) *Let $\langle P_0, Q_0\rangle$ and $\langle P_1, Q_1\rangle$ be pairs of predicates with arities n and $n+m$. Then we define $\langle P_0, Q_0\rangle \leq_R \langle P_1, Q_1\rangle$ as the conjunction of the following three formulas.*

1. $\forall \mathbf{x}.\, P_0(\mathbf{x}) \to P_1(\mathbf{x})$
2. $\forall \mathbf{x}, \mathbf{y}.\, Q_0(\mathbf{x}, \mathbf{y}) \to Q_1(\mathbf{x}, \mathbf{y})$
3. $\forall \mathbf{x}, \mathbf{y}.\, P_0(\mathbf{x}) \to Q_1(\mathbf{x}, \mathbf{y}) \to Q_0(\mathbf{x}, \mathbf{y})$

where $\mathbf{x}$ ($\mathbf{y}$) are sequences of n (m) distinct variables.

The first two conditions are the same as the conditions for the natural order $\leq_N$. The difference arises in the third one. This condition together with the second condition says, for any $\mathbf{x}$ satisfying $P_0(\mathbf{x})$, the predicates $Q_0(\mathbf{x}, \mathbf{y})$ and $Q_1(\mathbf{x}, \mathbf{y})$ are equivalent. In other words, Q_1 is larger than Q_0 only outside of the domain (defined by P_0). We call this order a refined order or an R-order, in short. We say that the second predicate Q_i depends on the first predicate P_i.

We can extend the R-order to the case of more than 2 predicates, for instance, Q_i depends on P_i, and R_i depends on Q_i. To formalize a general scheme, we introduce

[b] This relation is also an order *modulo logical equivalence*.

3.1. Monotone inductive definition

Tatsuta[21] introduced into Beeson's **EON**[13] a mechanism of monotone inductive definitions. Then he defined a $\mathtt{q}$-realizability interpretation, and proved its soundness for a restricted version. We first reformulate his results using our theory **BT** instead of **EON**.

Definition 3 (Natural order between abstracts) *Let P_0, P_1 be n-ary abstracts, and $\mathbf{x}$ be a list of n distinct variables. Then we define*

$$P_0 \leq_N P_1 \quad \stackrel{\triangle}{=} \quad \forall \mathbf{x}.P_0(\mathbf{x}) \to P_1(\mathbf{x})$$

This relation is an order *modulo logical equivalence*, namely we identify two predicates which are logically equivalent. The subscript N indicates this order is a natural one.

Definition 4 (Monotonicity wrt. $\leq_N$) *Let A be an n-ary abstract, which possibly contains an n-ary predicate variable X. Then we define*

$$\mathbf{MONO}(A; X) \quad \stackrel{\triangle}{=} \quad X_0 \leq_N X_1 \ \to \ A_X[X_0] \leq_N A_X[X_1]$$

where X_0 and X_1 are fresh n-ary predicate variables.

We often omit X in $\mathbf{MONO}(A; X)$. We say A is monotone in X if $\mathbf{MONO}(A; X)$ holds.

The mechanism of *monotone inductive definitions* (**MID**, in short) is that, for any A satisfying $\mathbf{MONO}(A; X)$, we have the least solution of $\forall \mathbf{x}.(A(\mathbf{x}) \leftrightarrow X(\mathbf{x}))$. We denote this solution as $\mu X.A$. Since there was a technical difficulty in attaching realizers to the full **MID** in a first order theory, Tatsuta formulated an additional condition called **MONO-Q**, and allowed **MID** only when $\mathbf{MONO\text{-}Q}(A; X)$ is satisfied. Roughly speaking, $\mathbf{MONO\text{-}Q}(A; X)$ is the condition that the relation "e realizes $A(\mathbf{x})$" is monotone in the relation "e realizes $X(\mathbf{x})$". Since $\mathbf{MONO}(A; X)$ does not subsume $\mathbf{MONO\text{-}Q}(A; X)$, this is a proper restriction.

The rules for $\mathbf{MID}_0$, a restricted version of **MID**, are as follows.

$$\frac{\mathbf{MONO}(A) \quad \mathbf{MONO\text{-}Q}(A)}{\forall \mathbf{x}.\ A_X[\mu X.A](\mathbf{x}) \leftrightarrow (\mu X.A)(\mathbf{x})} \ (\mu\text{-eq})$$

$$\frac{\mathbf{MONO}(A) \quad \mathbf{MONO\text{-}Q}(A) \quad A_X[C] \leq_N C}{\mu X.A \leq_N C} \ (\mu\text{-ind})$$

where C is an arbitrary n-ary abstract. The first rule states that $\mu X.A$ is a solution of $\forall \mathbf{x}.\ A(\mathbf{x}) \leftrightarrow X(\mathbf{x})$. The second rule states that $\mu X.A$ is a least solution. Tatsuta gave a realizability interpretation of the theory with $\mathbf{MID}_0$ in the theory with **MID**, and proved its soundness.

MID and $\mathbf{MID}_0$ can be extended to a simultaneous inductive definition straightforwardly.

Remarks

If a formula F is proved from a set of formulas $A_1, \cdots, A_n$ in a theory $\mathcal{T}$, we will write it as $A_1, \cdots, A_n \vdash_{\mathcal{T}} F$.

3. Inductive Definitions

In this section, we present an extended mechanism of simultaneous inductive definition, which plays a central role in our theory.

As we explained in Section 1, we need a simultaneous definition of two predicates `Prop` and `Proves`. The definition should allow negative occurrences of predicate variables in the template[a], which means the template may sometimes be non-monotonic. This kind of definitions is not legal in conventional formal theories for positive inductive definitions.

However, the intuitive meaning of the pair $\langle \mathtt{Prop}, \mathtt{Proves} \rangle$ has some kind of monotonicity; $\mathtt{Prop}(A \dot{\wedge} B)$ is defined by using $\mathtt{Prop}(A)$ and $\mathtt{Prop}(B)$, which are already defined. $\mathtt{Prop}(A \dot{\rightarrow} B)$ is defined by using $\mathtt{Prop}(A)$, $\mathtt{Prop}(B)$, and $\mathtt{Proves}(x, A)$. Since the set of those r satisfying $\mathtt{Proves}(r, A)$ is defined at the time of defining $\mathtt{Prop}(A)$ and will never be changed after defined, we may consider that $\mathtt{Prop}(A \dot{\rightarrow} B)$ is defined by using already defined concepts only.

To formalize this idea, we define a refined order $\leq_R$ on a pair of unary and binary predicates $\langle P, Q \rangle$ as follows:

$$\begin{aligned}\langle P_0, Q_0 \rangle \leq_R \langle P_1, Q_1 \rangle \stackrel{\triangle}{=} & \forall x.\ (P_0(x) \rightarrow P_1(x)) \ \wedge \ \forall x, y.\ (Q_0(x,y) \rightarrow Q_1(x,y)) \\ & \wedge \forall x.\ (P_0(x) \rightarrow \forall y.(Q_0(x,y) \leftrightarrow Q_1(x,y)))\end{aligned}$$

It is essential to have two predicates which have an *overlapping* argument. The second predicate Q_1 can be larger than Q_0 only outside of the domain of P_0, which means that those y satisfying $Q(x,y)$ is determined at the time $P(x)$ becomes true. With respect to $\leq_R$, the naive definition of $\langle \mathtt{Prop}, \mathtt{Proves} \rangle$ becomes monotone, and we can regard it as a legal inductive definition. We call a monotone inductive definition wrt $\leq_R$ a *half-monotone* inductive definition following Allen's term *half-positive*[17].

Aczel[14] used a similar order $\leq_A$ in his semantical framework. For two sets S and T with $T \subset S$, the order $\leq_A$ is defined as follows:

$$\langle S_0, T_0 \rangle \leq_A \langle S_1, T_1 \rangle \stackrel{\triangle}{=} S_0 \subset S_1 \wedge \ \forall x \in S_0.\ (x \in T_0 \leftrightarrow x \in T_1)$$

It is easily seen that $\leq_A$ is a special case of $\leq_R$; For a pair of sets $\langle S_i, T_i \rangle$, we define $P_i(x) \stackrel{\triangle}{=} x \in S_i$ and $Q_i(x,y) \stackrel{\triangle}{=} x \in T_i \wedge y = 0$. Then $\langle S_0, T_0 \rangle \leq_A \langle S_1, T_1 \rangle$ are equivalent to $\langle P_0, Q_0 \rangle \leq_R \langle P_1, Q_1 \rangle$ provided that $T_i \subset S_i$ (for $i = 0, 1$) holds. Hence, our order can be regarded as a generalization of Aczel's.

In this section, we first review Tatsuta's theory and realizability interpretation for ordinary monotone inductive definition in a context of a type-free first-order theory. We then introduce a half-monotone inductive definition and construct a model of the extended theory. A realizability interpretation and its soundness proof will be given in the next section.

[a] We call the defining formula in an inductive definition a *template*.

We then define formulas and abstracts. We assume, for each natural number n, there is a countably infinite set of predicate variables with arity n. Predicate variables are used for inductive definition. Since they are not quantified by $\forall$ or $\exists$, **BT** is a first-order theory.

Definition 2 (Formula and n-ary Abstract)

$$\begin{array}{rcl} F & ::= & \bot \mid t_1 = t_2 \mid t\downarrow \\ & \mid & F_1 \wedge F_2 \mid F_1 \vee F_2 \mid F_1 \to F_2 \\ & \mid & \forall x.F \mid \exists x.F \\ & \mid & A^n(t_1, \cdots, t_n) \\ A^0 & ::= & F \\ A^n & := & X^n \\ & \mid & \lambda x.A^{n-1} \\ & \mid & \mu X^n.A^n \\ & & (for\ each\ n > 0) \end{array}$$

where x, t and X^n are meta variables for a variable, a term, and an n-ary predicate variable.

A formula in the form of $\bot$, $t_1 = t_2$, or $t\downarrow$ is called *atomic*. $a\downarrow$ means that the term a has a canonical form. The formula $A^n(t_1, \cdots, t_n)$ represents the application of n terms $t_1, \cdots, t_n$ to an abstract A^n. The superscript n for predicate variables and abstracts is often omitted. We sometimes call an abstract a *predicate*. We write a list of n distinct variables $x_1, \cdots, x_n$ as $\mathbf{x}$, and a list of n terms $t_1, \cdots, t_n$ as $\mathbf{t}$. Similarly, the formula $A(t_1, \cdots, t_n)$ is abbreviated as $A(\mathbf{t})$. $\neg A$ and $A \leftrightarrow B$ are abbreviations for $A \to \bot$ and $(A \to B) \wedge (B \to A)$, respectively. The formula $A_X[B]$ represents the formula A with B substituted for X. We often omit the subscript X if it is apparent from the context. It is also written as $A\{X := B\}$.

In the following, we will use meta-variables x, y, z, w for variables, s, t, u for terms, A, B, F, G for formulas and abstracts, and X, Y for predicate variables. The precedence of connectives is in the order $\wedge, \vee, \to, \forall, \exists$ (the former is stronger). We also assume $\to$ is right associative, namely $A \to B \to C$ means $A \to (B \to C)$.

The formal system of **BT** is given in the natural deduction style. We have the following three classes of axioms and inference rules:

- Axioms and Rules for terms
- Rules for equality
- Rules for logical connectives

These axioms and rules are quite standard for first-order intuitionistic logic, and therefore omitted. We do not adopt a logic of partial terms, so quantifiers range over arbitrary terms, not necessarily canonical nor normal terms.

Martin-Löf's type theory can be interpreted. We also show that we can improve the efficiency of programs by defining a refined provability relation. Our theory enjoys a sound realizability interpretation, so that we have the program extraction theorem. By this theorem, we can extract programs from proofs of formalized metatheorems.

This paper is organized as follows. Section 2 gives our basic theory **BT**. Section 3 introduces our inductive definition mechanisms, **HMID** and $\mathbf{HMID}_0$ into **BT**, and gives its model. Section 4 presents a realizability interpretation, and proves soundness for a restricted version $\mathbf{BT}{+}\mathbf{HMID}_0$. Section 5 formalizes provability relations in our theory. We also show that program improvement is possible in $\mathbf{BT}{+}\mathbf{HMID}_0$. Section 6 interprets the Logical Theory of Constructions and briefly mentions the interpretation of Martin-Löf's type theory in $\mathbf{BT}{+}\mathbf{HMID}_0$. Section 7 compares our work with Allen's and Dybjer's works, and gives concluding remarks.

2. The basic theory

We define our basic theory **BT** in this section.

BT is an intuitionistic first-order theory for computation. Although our final goal is to formalize Frege structures[14], Reflective Proof Theory[22] and similar theories, we shall give our basic results in the framework of ordinary first-order logic, since it is easier to understand.

The definition of terms in our language is a slight modification of Sato's[22]. Terms are essentially type-free λ-terms with α and β conversions. We adopt the call-by-name semantics, but the material of this paper can easily be applied to the call-by-value semantics.

We assume that there is a countably infinite set of variables and a finite set of constants, and that `nil`, `true`, and `false` are constants. We also assume that a unary function symbol $\mathbf{c?}$ is uniquely associated with each constant $\mathbf{c}$.

Definition 1 (Term)

$$
\begin{array}{rcl}
t & ::= & x \\
& | & \mathbf{c} \mid \mathbf{c?}(t) \\
& | & \langle t_1, t_2 \rangle \mid \texttt{car}(t) \mid \texttt{cdr}(t) \mid \texttt{pair?}(t) \\
& | & \lambda x.t \mid t_1(t_2) \mid \texttt{fun?}(t) \\
& | & \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \texttt{ fi}
\end{array}
$$

where x is a variable and $\mathbf{c}$ is a constant.

Terms $\langle t_1, t_2 \rangle$, $\texttt{car}(t)$, and $\texttt{cdr}(t)$ correspond to `cons`, `car`, and `cdr` in Lisp. We use the standard abbreviation for lists; $\langle t_1 \rangle$ for $\langle t_1, \texttt{nil} \rangle$, and $\langle t_1\ t_2 \rangle$ for $\langle t_1, \langle t_2, \texttt{nil} \rangle \rangle$ and so on. Terms $\lambda x.t$, $t(t)$ and $(\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \texttt{ fi})$ are as usual. Terms of the form of $\mathbf{c}$, $\langle t_1, t_2 \rangle$, or $\lambda x.t$ are called *canonical*. For a canonical term t, the terms $\mathbf{c?}(t)$, $\texttt{pair?}(t)$ and $\texttt{fun?}(t)$ are equal to `true` if they are in either form of $\mathbf{c?}(\mathbf{c})$, $\texttt{pair?}(\langle s, u \rangle)$ or $\texttt{fun?}(\lambda x.s)$, and equal to `false` otherwise. We assume a call-by-name evaluation mechanism; for example, $\texttt{car}(\langle \texttt{nil}, t \rangle)$ is equal to `nil` regardless of t. A term in the form of $a_x[t]$ denotes the result of substitution as usual.

be *monotone*. However, in the above formulation, an occurrence of $\mathtt{Proves}(e, u)$ appears in the left side of $\rightarrow$, which means that it is a negative occurrence. This kind of inductive definitions does not fall in an ordinary scheme of *positive* inductive definitions.

Several researchers have attacked this problem in different contexts.

Beeson[13] used two kinds of techniques; the first one uses the ordinal numbers in classical set theory. The other one simultaneously defines a pair of the predicates $\mathtt{Proves}$ and its negation $\overline{\mathtt{Proves}}$ by a monotone inductive definition, and then $\neg\mathtt{Proves}(x, y) \leftrightarrow \overline{\mathtt{Proves}}$ is proved using the law of excluded middle. Both techniques rely on classical logic.

Aczel[14] proposed Frege structures. Frege structures have two basic notions, "a term a is a proposition" and "a term a is a true proposition". In order to construct Frege structures in set theory, he encountered similar difficulty. As a solution, he introduced a new order between a pair of predicates. With respect to this order, the pair of defining formulas of the two notions above becomes monotone, therefore the least fixpoint exists. Although his metatheory was classical, he stated that the construction might be considered constructive. Hayashi and Nakano[4] used a similar order to construct models for their theory *PX*. Our method is similar to their works in that we regard our inductive definitions as monotone by changing the definition of the order. What is new in our theory is that we give a general form of such a style of inductive definitions, and that we do everything in a constructive framework in the sense that our theory has a sound realizability interpretation, and enjoys the program extraction theorem. In this paper, we will interpret a formalized version of Frege structures (the Logical Theory of Constructions[15]) in our theory.

Allen[16,17] gave a type-free interpretation for Martin-Löf's type theory. He encountered a similar situation as ours, and called the necessary scheme of inductive definitions *half-positive*. By using an inductive definition for higher order predicates, he was able to use ordinary (strictly positive) inductive definitions to interpret Martin-Löf's type theory. His arguments can be understood by classical set theorists as well as constructivists who accept monotone inductive definitions. Although his motivation was different from ours, we shall compare his work and ours in Section 7. Smith[18] formalized Allen's interpretation in a Martin-Löf's type theory plus recursive types.

Dybjer[19,20] presented a general formulation of a recursive-induction mechanism so that the universe hierarchy becomes definable in Martin-Löf-style type theory. The motivation of our theory is similar to his approach, and we shall also compare his work and ours in Section 7.

In this paper we present a general mechanism of inductive definitions by which one can define the provability relation and similar notions naturally. The mechanism is based on a form of a monotone inductive definition with a refined order between predicates. We call this mechanism *half-monotone* inductive definitions. We define a realizability interpretation to our theory, and prove its soundness by following Tatsuta's work[21]. We shall demonstrate how various concepts can be defined in our theory; in particular, we show that the Logical Theory of Constructions and

inefficient program. Much research has been done on this topic; Subset Types[2,6], Separation of `Spec` and `Prop` types[8], and SUIT (Singleton, Union and Intersection Types)[9] in type theories, and Diamond suit (double negation)[4], and Extended projection[10] in type-free theories. These techniques introduce new types (in type theories) or new realizability interpretations (in type-free theories) by which we can eliminate redundant parts in programs. In other words, each of these systems gives a fixed, uniform way of program improvement. In order to introduce a new technique for improvement, they must re-define the whole system and re-prove its consistency (in type theories), or re-define the realizability interpretation and re-prove its soundness (in type-free theories). These tasks belong to meta-theories, and go beyond the original theory.

Our aim is to formalize various program improvement techniques in a single framework. Namely, we want to have a mechanism to define and reason about the relation "a term a is a program (proof) of a type (proposition) A". Since this is a metatheoretic notion, our theory should include a certain kind of reflection. If such a theory is formulated, we can add a new optimization technique to the system by re-defining the relation, and prove, for instance, equivalence of the old and new definitions. Moreover, if the metatheory is also constructive in nature, we can extract an optimization program from the proof in the metatheory. The reflection mechanism is quite useful in proof/program development, as pointed out by Allen et al[11].

However, the natural definition of these relations leads us outside the realm of *positive* inductive definitions as shown below. We call a pair of the following two relations a *provability relation*.

A is a proposition (written as $\mathtt{Prop}(A)$)
a is a proof of A (written as $\mathtt{Proves}(a,A)$)

In Martin-Löf's type theory[12], these relations correspond to the judgements $A\ Set$ and $a \in A$. In a usual first order logic, they correspond to a metamathematical statement "A is a formula" and a formula "a realizes A". If we formulate $\mathtt{Prop}(x)$ and $\mathtt{Proves}(x,y)$ naturally, they look like:

$$
\begin{array}{rcl}
\mathtt{Prop}(y) & \leftrightarrow & \cdots \\
 & & \exists u.\exists v.\ \mathtt{Prop}(u) \ \wedge\ y = (u \dot{\rightarrow} v) \ \wedge\ \forall e.\ (\mathtt{Proves}(e,u) \rightarrow \mathtt{Prop}(v)) \\
 & & \cdots \\
\mathtt{Proves}(x,y) & \leftrightarrow & \cdots \\
 & & \exists u.\exists v.\ \mathtt{Prop}(u) \ \wedge\ y = (u \dot{\rightarrow} v) \\
 & & \qquad \wedge\ \forall e.\ (\mathtt{Proves}(e,u) \rightarrow \mathtt{Proves}(x(e),v)) \\
 & & \cdots
\end{array}
$$

where $\dot{\rightarrow}$ represents a constant corresponding to logical implication, and $\leftrightarrow$ represents logical equivalence. In order to regard these clauses as an instance of simultaneous inductive definitions for two predicates, the right hand side of $\leftrightarrow$ should

International Journal of Foundations of Computer Science
© World Scientific Publishing Company

A TYPE-FREE THEORY OF HALF-MONOTONE INDUCTIVE DEFINITIONS

YUKIYOSHI KAMEYAMA
Research Institute of Electrical Communication,
Tohoku University, Sendai, 980-77 JAPAN

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

This paper studies an extension of inductive definitions in the context of a type-free theory. It is a kind of simultaneous inductive definition of two predicates where the defining formulas are monotone with respect to the first predicate, but not monotone with respect to the second predicate. We call this inductive definition *half-monotone* in analogy of Allen's term *half-positive*.
We can regard this definition as a variant of monotone inductive definitions by introducing a refined order between tuples of predicates. We give a general theory for half-monotone inductive definitions in a type-free first-order logic. We then give a realizability interpretation to our theory, and prove its soundness by extending Tatsuta's technique. The mechanism of half-monotone inductive definitions is shown to be useful in interpreting many theories, including the Logical Theory of Constructions, and Martin-Löf's Type Theory. We can also formalize the provability relation "a term p is a proof of a proposition P" naturally. As an application of this formalization, several techniques of program/proof-improvement can be formalized in our theory, and we can make use of this fact to develop programs in the paradigm of Constructive Programming. A characteristic point of our approach is that we can extract an optimization program since our theory enjoys the program extraction theorem.

Keywords: Constructive programming, Reflection, Universe, Inductive definition, Realizability, Redundancy-free program

1. Introduction

Constructive programming is a method of program development based on constructive logic in which program extraction from logical specifications and the correctness of programs are guaranteed[1,2,3,4,5,6,7]. If a specification formula of the form $\forall x.\exists y.A(x,y)$ is proved in constructive logic, we can then extract a program f from the proof, and at the same time, we have a proof of $\forall x.A(x,f(x))$, which guarantees the correctness of the program with respect to the specification formula.

An important problem in constructive programming is that extracted programs often contains redundant parts. Namely, a naive extraction usually produces an