ICOT Technical Report: TR-0802

TR-0802

Parallel Process Synthesis from Proofs on Logic μ

by
H. Kawada, Y. Sato & M. Fujita

September, 1992

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Parallel Process Synthesis from Proofs on Logic μ

Hideji Kawata Yosuke Sato Masayuki Fujita
Institute for New Generation Computer Technology
1-4-28 Mita, Minato-ku, Tokyo 108, Japan
kawata@icot.or.jp ysato@icot.or.jp mfujita@icot.or.jp

The technique in Howard (1984), Martin-Löf (1982), Sato (1986) and Hayashi & Nakano (1987) to generate programs from constructive proofs has many useful points. For instance, we can synthesize a consistent program if the proof of the formula, as the specification, is consistent. In this technique, however, we cannot synthesize a parallel process (like CCS (Milner (1989)) or CSP (Hoare (1985))) by which computers communicate with the outside world in a natural manner, because the definition of "program" in this technique is only "λ term". In this paper, we propose a new logic μ, and a technique for generating a parallel process by which computers communicate with the outside world from the proof of this logic.

1 Introduction

There has been some research (Howard (1984), Martin-Löf (1982), Sato (1986), and Hayashi & Nakano (1987)) into program synthesis from constructive proofs. In this method, an interpretation of formulas is defined as a set of programs, and a translation method, from the consistent proof of the formula into a program that satisfies the interpretation, is shown. Therefore, we can identify the formula as the specification of the program, the proof as the program, and proof checking as program checking. Thus, this method has many useful points, however, the definition of "program" in this method is only "λ Term (function)". This make it, difficult to synthesize a program as a parallel process by which computers communicate with the outside world. In this paper, we propose a method to synthesize "program" as a parallel process like CCS (Milner (1989)) from constructive proofs. Section 2 gives the parallel functional language "Program Term" and some termination properties. In section 3, we discuss a method to express a process using a formula. Section 4 proposes a new logic μ to synthesize the process from constructive proofs. Section 5 defines a realizability of logic μ using Program Term, and shows a

program synthesis method from the proof of μ. Section 6 shows an example of program synthesis.

2 Program Term

Program Term is a parallel and functional language. This language can express communication with an outside world.

Let U, U_v and U_f be pairwise disjoint sets, where each element of U is an individual constant, each element of U_v is an individual variable, and each element of U_f is a function constant. We use h, i, j, k, m and n for natural numbers, c and d for individual constants, s, t, u, v, w, x, y and z for individual variables, f_n, g_n and h_n for n-ary function constants, a and b for objects, ϵ, A, B, C and D for Program Terms, and X, Y and Z for Program Term variables.

2.1 Syntax

Definition 2.1 (Object, Program Term)
We define objects and Program Term as follows.

- object $a ::= x|c|f_{(n)}a_1...a_n$
- Program Term $A ::= a|nop|Z|proj(n)|r|l|if(A = B)|\lambda_x.A|\lambda_Z.A|AB$
  $|select(A, B, C)|(\Pi, A)|(A, (B, C))|(in, A)|(nonin, A)|(out, A, B)$
  $|(para, A, B)|(seq, A, B)|(nondet, A, B)|(nonout, A, B)|\mu.A$

We assume that seq, $para$, $proj$, in, $nondet$, $nonin$, out, $nonout$, r, l, if, nop, $select$, λ, Π and μ can be distinguished from individual constants, individual variables, and function constants. We submit the following syntax sugars.

$$\lambda_{x_1,...,x_n}.A \stackrel{def}{\equiv} \lambda_{x_1}....\lambda_{x_n}.A \qquad \lambda_{Z_1,...,Z_n}.A \stackrel{def}{\equiv} \lambda_{Z_1}....\lambda_{Z_n}.A$$

and three occurrences of x.

Definition 2.2 (Object free variable, Object bound variable)
An occurrence of an individual variable x in Program Term A is said to be an object bound variable if there exists an A's subterm $\lambda_x.B$ such that this occurrence is in B. An individual variable x in Program Term A is said to be an object free variable if all occurrences of x in A are not object bound variables.

Definition 2.3 (Program Term bound variable, Program Term free variable)
An occurrence of a Program Term variable Z in Program Term A is said to be a Program Term bound variable if there exists an A's subterm $\lambda_Z.B$ such that this occurrence is in B. A Program Term variable Z in Program Term A is said to be an object free variable if all occurrences of Z in A are not Program Term bound variables.

Definition 2.4 (Executable Program Term)
A Program Term is said to be an executable Program Term if this Program Term has no Program Term free variable and no object free variable.

2.2 Operational Semantics

In this section, we define the operational semantics of the executable Program Term. First we define N-calculation, which is used to define the operational semantics.

Definition 2.5 (N-calculation, Normal Form)
Let D be a Program Term. N-calculation is defined as outermost rewriting by the following rules. If N-calculation of D terminates, then the result of N-calculation D' is called the Normal Form of D.

1. If there is an occurrence of the form $select(r, B, C)$, then replace this occurrence with B.
2. If there is an occurrence of the form $select(l, B, C)$, then replace this occurrence with C.
3. If there is an occurrence of the form $(\lambda_x.A(x))a$, then replace this occurrence with $A(a)$.
4. If there is an occurrence of the form $(\lambda_Z.A(Z))B$, then replace this occurrence with $A(B)$.
5. If there is an occurrence of the form $proj(j)(A_1, ..., A_i)$ and $j \leq i$, then replace this occurrence with $A(j)$.
6. If there is an occurrence of the form $proj(j)(A_1, ..., A_i)$ and $i < j$, then replace this occurrence with nop.
7. If there is an occurrence of the form $f_n(c_1, ..., c_n)$, then replace this occurrence with the result of calculation in $f_n(c_1, ..., c_n)$.
8. If there is an occurrence of the form $if(A = B)$, no object free variables or Program Term free variables occur in A and B, and no rewrite rules can be applied to A and B, then, if $A \equiv B$ then replace this occurrence with l else replace with r.
9. If there is an occurrence of the form $(\mu.A)a_1...a_n$, and this occurrence is not in B and C where $select(E, B, C)$ is an arbitrary occurrence, then replace this occurrence with $(A(\mu.A))a_1...a_n$.

Lemma 2.1 (Church Rosser Property)
Church Rosser property holds for N-calculation.

Definition 2.6 (Strongly Normalizable)
A Program Term A is called SN(Strongly Normalizable) if every N-calculation sequence is finite.

Definition 2.7 (Alpha Equal)
Let A and B be Program Terms.

- If a term of the form $\lambda_x.C$ occurs in A, and there exists an object variable y such that y does not occur in C, then, we write

$$A =_\alpha [\lambda_y.[y/x]C/\lambda_x.C]A.$$

- If a term of the form $\mu.\lambda_{Z_1}.C$ occurs in A, and there exists a Program Term variable Z_2 such that Z_2 does not occur in C, then, we write

$$A =_\alpha [\mu.\lambda_{Z_2}.[Z_2/Z_1]C/\mu.\lambda_{Z_1}.C]A.$$

If there exists a Program Term sequence $A_1, ..., A_n$ such that

$$A =_\alpha A_1,\ A_1 =_\alpha A_2,\ \ldots,\ A_n =_\alpha B,$$

then we write

$$A =_{\alpha*} B.$$

Definition 2.8 (Equality)
Let A and B be Program Terms, A' be a Normal Form of A, and B' be a Normal Form of B. Then,

$$A = B \overset{def}{\Longleftrightarrow} A' =_{\alpha*} B'.$$

First, we show the intuitive meaning of the execution of Program Term. Let P be an executable Program Term.

1. Execute P' which is the Normal Form of P as step 2.
2. If the Program Term has one of the forms that follow, then execute it, otherwise execution ends.
   - $(l, (A, B))$ ··· Execute A.
   - $(r, (A, B))$ ··· Execute B.
   - (seq, A, B) ··· Execute A and if A terminates then execute B.
   - $(para, A, B)$ ··· Execute A and B in parallel.
   - $(nondet, A, B)$ ··· Execute A or B(the selection is non-deterministic).
   - (Π, A) ··· Terminate.
   - (in, A) ··· Input a and if Aa has Normal Form A' then Execute A'.
   - (out, a, A) ··· Output a, then execute A.
   - $(nonout, B, A)$ ··· Execute A.

The operational semantics of Program Term are shown by the following definition.

Definition 2.9 (Operational Semantics of Program Term)
The operational semantics of Program Term are defined using a transition tree. $A \xrightarrow{t} A'$ means if A' has a normal form, then there is a transition from A to the normal form of A' by the action t. Action $in(b)$ means the action "input b", action $out(b)$ means the action "output b", action τ means the non-observable action or the internal action, and action a shows one of the above actions.

$$\frac{}{(l, (A, B)) \xrightarrow{\tau} A} \qquad \frac{}{(r, (A, B)) \xrightarrow{\tau} B}$$

$$\frac{}{(para, A, B) \xrightarrow{\tau} ((para, A, B), \tau)} \qquad \frac{}{(para, A, B) \xrightarrow{\tau} ((para, A, B), l)}$$

$$\frac{A \xrightarrow{a} A'}{((para, A, B), l) \xrightarrow{a} (para, A', B)} \qquad \frac{B \xrightarrow{a} B'}{((para, A, B), r) \xrightarrow{a} (para, A, B')}$$

$$\frac{}{(para, \bot, B) \xrightarrow{\tau} B} \qquad \frac{}{(para, A, \bot) \xrightarrow{\tau} A}$$

$$\frac{A \xrightarrow{a} A'}{(seq, A, B) \xrightarrow{a} (seq, A', B)} \qquad \frac{}{(seq, \bot, B) \xrightarrow{\tau} B}$$

$$\frac{}{(nondet, A, B) \xrightarrow{\tau} A} \qquad \frac{}{(nondet, A, B) \xrightarrow{\tau} B}$$

$$\frac{}{(in, A) \xrightarrow{in(b)} Ab} \ (bel') \qquad \frac{}{(out, b, A) \xrightarrow{out(b)} A} \ (bel')$$

$$\frac{}{(nonout, b, A) \xrightarrow{\tau} A} \qquad \frac{}{P \xrightarrow{\tau} \bot}$$

(P is not one of the above forms.)

2.3 Some Termination Properties on Program Term

We discuss some termination properties of the parallel process by which computers communicate with the outside world, using Program Term. In conclusion, we propose TDC (Termination by Delayed Control) as the property that the correct program must satisfy.

Definition 2.10 (Strong terminal node)
Let A be a Program Term and ℓ be the transition tree of A. We define a strong terminal node of ℓ as follows.

- If a node α is $\bot$, then α is a strong terminal node.
- If every node which can be reached by any action from α is a strong terminal node, then α is a strong terminal node.

We write $TC(\alpha)$ if α is a strong terminal node.

Definition 2.11 (Termination without Control)
Let A be a Program Term and ℓ be the transition tree of A. If the beginning node "A" of ℓ is a strong terminal node, then we say that A is TC (Termination without Control).

TC is an extension of the property "Strongly Normalizable" in the Term Rewriting System and the Abstract Rewriting System. Intuitively, "A Program Term B is TC" means "You can halt A by giving arbitrary finite input sequence." It seems simple, but some examples, like "editor" and "Operating System", do not satisfy TC. TC is too strong property to fix correct programs.

Definition 2.12 (Weak Terminal Node)
Let A be a Program Term and ℓ be the transition tree of A. We define a weak terminal node of ℓ as follows.

- If a node α is $\perp$, then α is a weak terminal node.
- If α has no transitions with an input action, and α cannot reach a non weak terminal node by any action, then α is a weak terminal node.
- If α has some transitions with an input action, and α cannot reach a non weak terminal node by any non-observable action or by output action, and every node which can be reached from α by some input is a weak terminal node, then α is a weak terminal node.

We write $TIC'(\alpha)$ if α is a weak terminal node.

Definition 2.13 (Termination by Immediate Control)
Let A be a Program Term and ℓ be the transition tree of A. If the beginning node "A" of ℓ is a weak terminal node, then we say that A is TIC' (Termination by Immediate Control).

TIC' is an extension of the property "Weakly Normalizable" in the Term Rewriting System and the Abstract Rewriting System. Intuitively, "A Program Term B is TIC'" means "You can halt B by giving adequate finite input sequence according to the state of B." But it may be that "you cannot halt A at any cost after giving incorrect input sequence". Therefore, TIC' is too weak property to fix correct programs.

Definition 2.14 (Terminal Node)
Let A be a Program Term and ℓ be the transition tree of A. We define a terminal node of ℓ as follows.

- If every node that can be reached by any finite sequence of transitions from α is a weak terminal node, then α is a terminal node.

We write $TDC'(\alpha)$ if α is terminal node.

Definition 2.15 (Termination by Delayed Control)
Let A be a Program Term and ℓ be the transition tree of A. If the beginning node "A" of ℓ is a terminal node, then we say A is TDC (Termination by Delayed Control). We denote that A is TDC by $TDC(A)$.

Intuitively, "A Program Term B is TDC" means "Even after arbitrary inputs, you can halt A by giving adequate finite input sequence according to the state of B". This property is better than TIC, because we can always recover inconsistent inputs. "Editor"and "Operating System" satisfy this property. Therefore, we propose TDC as the property to fix correct programs.

Definition 2.16 (Function *tic*)
Let A be a Program Term, ℓ be the transition tree of A, and α be a node of ℓ. We define the function "tic" as follows.

- If α is TDC then $tic(\alpha) = \omega$.
- If α is not TIC then $tic(\alpha) = 0$.

- If α is TIC, and every node that can be reached by any input sequence which has length $n-1$ is TIC, and some nodes that can be reached by any input sequence which has length m are not TIC where $m < n$, then $tic(\alpha) = n$.

We use the following order.

- $n < \omega$. If $n <_{nat} m$, where $<_{nat}$ is the general order on Natural Number, then $n < m$.

We identify the Program Term A as the beginning node "A" of the transition tree of A.

Lemma 2.2 (Some Property of TDC)

1. *If $TDC(A)$ and $TDC(B)$ then*
   - $TDC((para, A, B))$, $TDC((seq, A, B))$, $TDC((nondet, A, B))$
2. *If $n \leq tic(A)$ and $tic(B)$ then*
   - $n \leq tic((para, A, B))$, $n \leq tic((seq, A, B))$, $n \leq tic((nondet, A, B))$

3 Expression of Process

We show the primitive actions of process on CCS as an example.

(a) $\alpha(x).P(x)$	...	Input x from the port α and execute $P(x)$.
(b) $\bar{\alpha}(a).P$	...	Output a to the port α and execute P.
(c) $P_1 \mid P_2$	...	Execute P_1 and P_2 in parallel.
(d) $P_1 + P_2$	...	Execute P_1 or P_2 non-deterministically.

In Program Term, (a) corresponds to $(in.A)$, (b) corresponds to (out, a, A), (c) corresponds to $(para, A, B)$, and (d) corresponds to $(nondet, A, B)$. Note that we do not use the concept of "port", in order to keep our discussion simpler.

In this section, we discuss a method to express these actions using formulas, and we show the basic idea for expressing them. First we show how to express (a) and (b), and then how to express (c) and (d). Finally we show the problem with the idea, and discuss a solution to such a problem.

3.1 Expressions of Input and Output

In the classical method of program generation, some interpretation of a formula is inductively defined on a set of programs. Intuitively, if a formula F is interpreted by a program e, then e satisfies the specification F. We try to change the interpretion in order to express (a) and (b).

The classical interpretation of $\forall x F(x)$ is,

- f, such that for all a, fa interprets $F(a)$.

To express the input action, we change the interpretion of $\forall x F(x)$ to

- (in, f), such that for all a, fa interprets $F(a)$.

Similarly, to express the output action, we change the interpretation of $\exists x F(x)$ to

- (out, a, f), such that fa interprets $F(a)$

Representing some examples of programs to use this interpretation, it is necessary to have another $\forall$ and another $\exists$ that show conditions without actions.

To distinguish them, we use the notations $\dot{\forall}$ and $\dot{\exists}$ to show action and use the notation $\forall$ and $\exists$ to show actions. This discussion is similar to the discussion on the classical interpretation in Hayashi & Nakano (1987) and Takayama (1989).

We showed the idea of interpretation to express the input action and the output action, however, there is a problem with this idea.

We discuss this problem in subsection 3.3.

3.2 Parallelism and Nondeterminism

Similarly for subsection 3.1, we try to change the interpretation in order to express (c) and (d).

In the classical interpretion, the interpretation of $F \wedge G$ is

- (f, g), such that f interprets F, and g interprets G.

(f, g) has information for executing f and g. Therefore, we can change the interpretation of this formula as follows.

1. $(para, f, g)$, such that f interprets F, and g interprets G.
2. (seq, f, g), such that f interprets F, and g interprets G.
3. $(nondet, f, g)$, such that f interprets F, and g interprets G.

We use the notations $F \wedge_s G$ for 1, and $F \wedge G$ for 2 and $F \wedge_n G$ for 3.

3.3 Problem and Extension of Logic

Recall that our aim is to establish a method to synthesize, from proofs, a parallel process by which computers communicate with the outside world. For such an aim, we need to be able to express the specification of the general program using formulas. But can we express the specification of "Operating System" using the idea shown in subsections 3.1 and 3.2 ? The answer is no. Because, since every formula has a finite length, and $\forall$ and $\exists$ cannot appear infinitely in every formula, we cannot express a specification of a program that has no upper bounds to its input and output. We solved this problem of extending the logic. Intuitively, we extend a constructive logic to be able to express formulas of infinite length. Of course, the expression of formulas is not as free as in Infinitary Logic. We only accept formulas that can be represented recursively. We use the notation of the recursive formula as follows.

$$(\mu.\lambda_{\Psi_{(n)},x_1,\ldots,x_n}.F(x_1,\ldots,x_n,\Psi_{(n)}a_{11}(x_1,\ldots,x_n)\ldots a_{1n}(x_1,\ldots,x_n),\\ \ldots,\Psi_{(n)}a_{m1}(x_1,\ldots,x_n)\ldots a_{mn}(x_1,\ldots,x_n)))b_1\ldots b_n$$

(We abbreviate the form $a_{ij}(x_1, ..., x_n)$ to a_{ij}.)

This means

$$P(b_1, ..., b_n) \Leftrightarrow F(b_1, ..., b_n, P(a_{11}(b_1, ..., b_n), ..., a_{1n}(b_1, ..., b_n)), \\ ..., P(a_{m1}(b_1, ..., b_n), ..., a_{mn}(b_1, ..., b_n))).$$

The truth value and inference rules of recursive formulas are defined so that every extraction program from the correct proof by them satisfies TDC. The precise definition of logic μ is shown in section 4.

4 Logic μ

In this section, we propose logic μ. Definitions of individual constants, individual variables, and function constants are the same as in section 2. Let U_p and U_{pv} be disjoint sets (U_p, U_{pv}, U, U_v, and U_f are pairwise disjoint sets), where each element of U_p is a predicate constant, and each element of U_{pv} is a predicate variable. We use $o_{(n)}$, $p_{(n)}$, and $q_{(n)}$ for n-ary predicate constants, $\Psi_{(n)}$ and $\Phi_{(n)}$ for n-ary predicate variables, F, G, H, I, J, K, and L for I-formulas or formulas, and δ and γ for sequents.

4.1 Syntax

Definition 4.1 (Predicate, I-formula, Formula)
We define object, predicate, I-formula, formula, and sequent as follows.

1. Object ····· $a ::= x | c | f_n(a_1, ..., a_n)$
2. Predicate ····· $P_{(n)} ::= p_{(n)} | \Psi_{(n)}$
3. I-formula ····· $F ::= P_{(n)}a_1...a_n | \Psi_{(n)}a_1...a_n | F_1 \wedge F_2 | F_1 \vee F_2 | F_1 \Rightarrow F_2 | \neg F | \forall x F | \exists x F | a \mid \\ \natural x(\Phi_{(n)}, \lambda_{x_1,...,x_n}.G, \mu.\lambda_{\Psi_{(n)},y_1,...,y_n}.L, F)](\mu.\lambda_{\Psi_{(n)},x_1,...,x_n}.L)a_1...a_n$

   Where,

   (a) L is an I-formula in which $\natural$ or any predicate free variable except $\Psi_{(n)}$ do not occur, and $\Psi_{(n)}$ does not occur in the negative part (refer to **definition** 4.5) in L.

   (b) If F is an I-formula in which $\natural$ occurs at least twice, and $\natural x.(t_1, t_2, t_3, t_4)$ and $\natural x.(s_1, s_2, s_3, s_4)$ are arbitrary subformulas of F, then the following relations must hold.

$$t_1 \equiv s_1 \quad t_2 \equiv s_2 \quad t_3 \equiv s_3$$

4. Formula ····· $G ::= P_{(n)}a_1...a_n | G_1 \wedge G_2 | G_1 \vee G_2 | G_1 \Rightarrow G_2 | \neg G | \forall x G | \exists x G | a \mid \mid \\ (\mu.\lambda_{\Psi_{(n)},x_1,...,x_n}.L)a_1...a_n$

   Where L is I-formula in which $\natural$ or any predicate free variable expept for $\Psi_{(n)}$ do not occur, and $\Psi_{(n)}$ does not occur in the negative part (refer to **definition** 4.5) in L.

5. Sequent ····· $\delta ::= F_1, .., F_n \vdash F$

   Note that we sometimes abbreviate the form $F_1, .., F_i$ to Γ and Δ.

Definition 4.2 ($\langle\ \rangle$)
Let A and F be arbitrary I-formulas, then

$$(A/\Psi)F \stackrel{def}{\equiv} \begin{cases} \text{The result of replacing } \Psi \text{ in } F \text{ with } A, \text{ except inside the I-formula} \\ \text{whose outermost logical symbol is } \natural. \end{cases}$$

We introduce the new logical symbol $\natural$. The intuitive meaning of $\natural$ is

$$\natural x(\Phi_{(n)}, \lambda_{x_1,\ldots,x_n}.F, \mu.\lambda_{\Psi_{(n)},y_1,\ldots,y_n}.G, H(x))$$
$$\Leftrightarrow (\forall x_1,\ldots x_n(F \Rightarrow \Phi_{(n)}x_1..x_n) \Rightarrow \forall x H(x)) \wedge \exists x.\langle \mu.\lambda_{\Psi_{(n)},y_1,\ldots,y_n}.G/\Phi_{(n)}\rangle H(x).$$

(Precisely speaking, as we do not define the "Natural-Elim" rule, "$\Rightarrow$" is not satisfied.)

We do not need to define "$\natural$" in logic, but it is needed when we define the realizer interpretation and realizer extraction rules.

Definition 4.3 (Bound Variable, Free Variable, etc)

- Let F be an I-formula or formula, and x be an object variable in F. Then, we say that an occurrence of x is an object bound variable if there exists a subformula G such that G has one of the following forms:

$$\forall x H, \exists x H, (\mu.\lambda_{\Psi_{(n)},x_1,\ldots,x_{i-1},x,x_{i+1},\ldots,x_n}.H)a_1\ldots a_n,$$
$$\natural x(\Phi, \lambda_{x_1,\ldots,x_{i-1},x,x_{i+1},\ldots,x_n}.H, \mu.\lambda_{\Phi_{(n)},y_1,\ldots,y_n}.I, J),$$
$$\natural x(\Phi, \lambda_{x_1,\ldots,x_n}.H, \mu.\lambda_{\Phi_{(n)},y_1,\ldots,y_{i-1},x,y_{i+1},y_n}.I, J).$$

- Let F be an I-formula or formula, and Ψ be a predicate variable in F. Then, we say that an occurrence of Ψ is a predicate bound variable if there exists a subformula G such that G has one of the following forms:

$$(\mu.\lambda_{\Psi,x_1,\ldots,x_n}.H)a_1\ldots a_n, \natural x(\Phi, \lambda_{x_1,\ldots,x_{i-1},x,x_{i+1},\ldots,x_n}.H, \mu.\lambda_{\Psi,y_1,\ldots,y_n}.I, J).$$

- Let F be an I-formula or formula, and x be an object variable in F. Then, we say that x is an object free variable in F if there exists an occurrence of x such that the occurrence is not the object bound variable.
- Let F be an I-formula or formula, and Ψ be a predicate variable in F. Then, we say that Ψ is a predicate free variable in F if there exists an occurrence of Ψ such that the occurrence is not the predicate bound variable.

Definition 4.4 (Negative Context, Positive Context, and Input Context)
We simultaneously define the set of negative contexts Neg, the set of positive contexts Pos, and the set of input contexts In.

Let F, G be I-Formulas, N be an arbitrary element of Neg, P be an arbitrary element of Pos, and I be an arbitrary element of In. Then Neg, Pos, and In are inductively generated by the following clauses:

- $*, F \wedge P, P \wedge F, P \vee F, F \vee P, N \Rightarrow F, F \Rightarrow P, \forall x P, \exists x P,$
  $(\mu.\lambda_{\Psi_{(n)},y_1,\ldots,y_n}.P)a_1\ldots a_n, \natural x(\Phi_{(n)}, \lambda_{x_1,\ldots,x_n}.F, \mu.\lambda_{\Psi_{(n)},y_1,\ldots,y_n}.G, P),$
  $\natural x(\Phi_{(n)}, \lambda_{x_1,\ldots,x_n}.P, \mu.\lambda_{\Psi_{(n)},y_1,\ldots,y_n}.G, F), \natural x(\Phi_{(n)}, \lambda_{x_1,\ldots,x_n}.G, \mu.\lambda_{\Psi_{(n)},y_1,\ldots,y_n}.P, F) \in Pos$

- $F \wedge N,\ N \wedge F,\ N \vee F,\ F \vee N,\ P \Rightarrow F,\ F \Rightarrow N,\ \forall x N,\ \exists x N,$
  $(\mu.\lambda_{\Phi_{(n)} y_1 \ldots y_n}.N) a_1 \ldots a_n,\ \natural x(\Phi_{(n)}, \lambda_{x_1 \ldots x_n}.F, \mu.\lambda_{\Psi_{(n)} y_1 \ldots y_n}.G, N),$
  $\natural x(\Phi_{(n)}, \lambda_{x_1 \ldots x_n}.N, \mu.\lambda_{\Phi_{(n)} y_1 \ldots y_n}.G, F),\ \natural x(\Phi_{(n)}, \lambda_{x_1 \ldots x_n}.G, \mu.\lambda_{\Psi_{(n)} y_1 \ldots y_n}.N, F) \in Neg$

- $*,\ F \wedge I,\ I \wedge F,\ I \vee F,\ I \vee P,\ F \Rightarrow I,\ \forall x I,\ \exists x I,\ \natural x(\Phi_{(n)}, \lambda_{x_1 \ldots x_n}.F, \mu.\lambda_{\Phi_{(n)} y_1 \ldots y_n}.G, I),\ \in In$

Let $Form$ be the set of all formulas. The set of wide negative contexts, the set of wide positive contexts, and the set of wide input contexts are defined as $Neg \cup Form$, $Pos \cup Form$, and $In \cup Form$, respectively.

Definition 4.5 (Negative Part, Positive Part)
Let F and I be arbitrary I-formulas. If there exists a negative (positive) context O such that $I \equiv [F/*]O$ holds, then we say that F occurs in the negative (positive) part of I.

Definition 4.6 (Input Part, Positive Part)
Let F and I be arbitrary I-formulas. If there exists an input context O such that $I \equiv [F/*]O$ holds, then we say that F occurs in the input part of I.

Definition 4.7 (Function)
We define four functions as follows.

- $\varrho(\Phi, F, G, H)$
  1. If no I-formulas of the form $\forall x I(x)$ occur in H's input part, then
     $$\varrho(\Phi, F, G, H) \stackrel{def}{\equiv} H.$$
  2. If some I formulas of the form $\forall x I(x)$ occur in H's input part, and J is the result of replacing the left-outermost I-formula of these I-formulas with $\natural x(\Phi_{(n)}, F, G, I(x))$, then $\varrho(\Phi, F, G, H) \stackrel{def}{\equiv} J$.
- $\varpi(\Phi, F, G, H)$
  1. If no I-formulas of the form $\forall x I(x)$ occur in H's input part, then,
     $$\varpi(\Phi, F, G, H) \stackrel{def}{\equiv} H.$$
  2. Otherwise, $\varpi(\Phi, F, G, H) \stackrel{def}{\equiv} \varpi(\Phi, F, G, \varrho(\Phi, F, G, H))$.
- $\varphi(H)$
  1. If no I-formulas of the form $\natural x(\Phi_{(n)}, F, G, I(x))$ occur in H's input part, then,
     $$\varphi(H) \stackrel{def}{\equiv} H.$$
  2. If some I-formulas of the form $\natural x(\Phi_{(n)}, F, G, I(x))$ occur in H's input part, and J is the result of replacing the left-outermost I-formula of these I-formulas with $\forall x I(x)$, then,
     $$\varphi(H) \stackrel{def}{\equiv} J.$$
- $\vartheta(H)$
  1. If no I formulas of the form $\natural x(\Phi_{(n)}, F, G, I(x))$ occur in H's input part, then,

$$\vartheta(H) \stackrel{def}{\equiv} H.$$

2. Otherwise, $\vartheta(H) \stackrel{def}{\equiv} \vartheta(\varphi(H))$.

4.2 Inference Rules

We propose inference rules of logic μ.

$\frac{}{\Gamma \vdash F}$ (Assumption) (where F is in Γ)

$\frac{\Gamma \vdash F \quad \Gamma \vdash G}{\Gamma \vdash F \wedge G}$ (And-Intro) $\qquad \frac{\Gamma \vdash F \wedge G}{\Gamma \vdash F}$ (And-Elim 1)

$\frac{\Gamma \vdash F \wedge G}{\Gamma \vdash G}$ (And-Elim 2) $\qquad \frac{\Gamma \vdash F}{\Gamma \vdash F \vee G}$ (Or-Intro 1)

$\frac{\Gamma \vdash F \vee G \quad \Gamma, F \vdash H \quad \Gamma, G \vdash H}{\Gamma \vdash H}$ (Or-Elim) $\qquad \frac{\Gamma \vdash G}{\Gamma \vdash F \vee G}$ (Or-Intro 2)

$\frac{\Gamma, F \vdash G}{\Gamma \vdash F \Rightarrow G}$ (Imp-Intro) $\qquad \frac{\Gamma \vdash F \Rightarrow G \quad \Gamma \vdash F}{\Gamma \vdash G}$ (Imp-Elim)

$\frac{\Gamma, y \downarrow \vdash F(y)}{\Gamma \vdash \forall x F(x)}$ (Univ-Intro) $\qquad \frac{\Gamma \vdash \forall x F(x) \quad \Gamma \vdash a \downarrow}{\Gamma \vdash F(a)}$ (Univ-Elim)

(where y does not occur in the lower sequent, and a is an arbitrary object.)

$\frac{\Gamma \vdash F(a) \quad \Gamma \vdash a \downarrow}{\Gamma \vdash \exists x F(x)}$ (Exists-Intro) $\qquad \frac{\Gamma \vdash \exists x F(x) \quad \Gamma, F(y) \vdash G}{\Gamma \vdash G}$ (Exists-Elim)

(where y does not occur in the lower sequent, and a is an arbitrary object.)

$$\frac{\Gamma, \forall y_1, ...y_n(< y_1, ..., y_n > <_{jn} < x_1, ..., x_n > \Rightarrow F(y_1, ..., y_n)) \vdash F(x_1, ..., x_n)}{\Gamma \vdash \forall x_1, ..., x_n F(x_1, ..., x_n)} \text{ (WF-Ind)}$$

(where $x_1, ..., x_n$ do not occur in the lower sequent.)

$\frac{\Gamma \vdash G(b_1, ..., b_n, I)}{\Gamma \vdash I b_1 ... b_n}$ (Mu-1) $\qquad \frac{\Gamma \vdash I b_1 ... b_n}{\Gamma \vdash G(b_1, ..., b_n, I)}$ (Mu-2)

(where $I \equiv \mu.\lambda_{\Psi_{(n)}, x_1, ..., x_n}.G(x_1, ..., x_n, \Psi)$)

$$\frac{\Gamma \vdash \exists x \langle I/\Phi \rangle K(x) \quad \Gamma, \forall x_1, ...x_n(H \Rightarrow \Phi_{(n)} x_1 ... x_n) \vdash \forall x K(x)}{\Gamma \vdash \natural x(\Phi, \lambda_{x_1, ..., x_n}.H, I, K(x))} \text{ (Natural-Intro)}$$

(where $I \equiv \mu.\lambda_{\Phi_{(n)}, x_1, ..., x_n}.G(x_1, ..., x_n, \Psi)$)

$$\frac{\Gamma, H(y_1,...,y_n) \vdash \varpi(\Phi, \lambda_{x_1,...,x_n}.H(x_1,...,x_n), I, G(y_1,...,y_n,\Phi)) \quad \Gamma \vdash H(a_1,...,a_n)}{\Gamma \vdash I a_1 ... a_n} \text{ (Mu-Intro)}$$

(where $H(y_1,...,y_n)$ has the form $\neg L(y_1,...,y_n)$, $I \equiv \mu.\lambda_{\Psi_{(n)},x_1,...,x_n}.G(x_1,...,x_n,\Psi)$, and Φ does not occur in the lowest secuent.)

$$\frac{\Gamma \vdash F \quad \Gamma \vdash \neg F}{\Gamma \vdash \bot} \text{ (Bot-Intro.neg-Elim)} \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash F} \text{ (Bot-Elim)}$$

$$\frac{\Gamma, F \vdash \bot}{\Gamma \vdash \neg F} \text{ (Neg-Intro)} \qquad \frac{}{\Gamma \vdash a = a} \text{ (eq 1)}$$

$$\frac{\Gamma \vdash a = b \quad \Gamma \vdash F(a)}{\Gamma \vdash F(b)} \text{ (eq 2)} \qquad \frac{\Gamma \vdash a_1 < a_2 \quad \Gamma \vdash a_2 < a_3}{\Gamma \vdash a_1 < a_3} \text{ (lc)}$$

Definition 4.8 (Provable Sequent)
If a sequent, which consists of formulas only, can be inferred by these inference rules, then we call it a provable sequent.

4.3 Axiom

In this section we define the axiom set, the axiom, and the inference rule of the axiom.

Definition 4.9
Let $\mathbf{U}$ be an arbitrary set, I_n be a set of well-founded partial orders on $\mathbf{U}^n$, $\mathbf{U}_f$ be a set of n ary partial functions on $\mathbf{U}^n$, and V be a function such that

1. $\forall x \epsilon U(V(x) \epsilon \mathbf{U}) \cup \{*\}$
   where $*$ is a new symbol which means "undefined".
2. $\forall x \epsilon U_{p(n)}(V(x) \subset \mathbf{U}^n)$
   where $U_{p(n)}$ is the the set of all n-ary predicate symbols which are elements of U_p.
3. $\forall x \epsilon U_{f(n)}(V(x) \epsilon \mathbf{U}_{f(n)})$
   where $U_{f(n)}$ is the set of all n-ary function symbols which are elements of U_f, and $\mathbf{U}_{f(n)}$ is the set of all n-ary functions which are elements of $\mathbf{U}_f$.
4. $V(f(a_1,...,a_n)) = V(f)(V(a_1),...,V(a_n))$
5. If $V(a_i) = *$ for some i $(0 < i < n+1)$, then $V(f)(V(a_1),...,V(a_n)) = *$.
6. $V(=) = \{< a,b > | V(a) = V(b)\}$
7. $V(\perp) = \phi$
8. $V(\downarrow) = \{a | V(a) \neq *\}$
9. For all i, n, there exists $\prec_i \epsilon I_n$ such that

$$V(<_{in}) = \{< a_1,...,a_n,b_1,...,b_n > | (< a_1,...,a_n > \prec_i < b_1,...,b_n >\}$$

Then we call $< \mathbf{U}, I_n, \mathbf{U}_f, V >$ the structure.

Definition 4.10 (Axiom Set)
Let $< \mathbf{U}, I_n, \mathbf{U}_f, V >$ be an arbitrary structure. Then

$$Ax(< \mathbf{U}, I_n, \mathbf{U}_f, V >) \stackrel{def}{\equiv} \{pa_1...a_n | < a_1, ..., a_n > \epsilon V(p), p \epsilon U_p \cup \bigcup_{i,n}\{<_{in}\}\}.$$

Definition 4.11 (Axiom)
Let $\mathcal{M}$ be a structure and S be a subset of $Ax(\mathcal{M})$, then we call S an axiom on $\mathcal{M}$.

Definition 4.12 (Inference Rule of Axiom)
Let $\mathcal{M}$ be a structure, and Ax be an axiom on $\mathcal{M}$. If a sequent, which consists of formulas, can be inferred by the inference rules of logic μ and the following inference rule, then we call it a provable sequent on the axiom Ax.

$\frac{}{\Gamma \vdash F}$ (Axiom) (where $F \epsilon Ax$)

After this, we denote $V(a)$ a easily.

Theorem 4.1 (Consistency of Logic μ)
Logic μ is consistent.

Proof. See **Appendix 2.** □

5 Realizability Interpretation

In this section, we introduce a realizability interpretation of logic μ, and program extraction rules. Using this interpretation, we can regard the formula in logic μ as the specification of programs. Moreover, by program extraction rules, we can regard proving the formula as programming the process.

5.1 Definition of the Realizability

We show a realizability interpretation of logic μ with some supporting symbols. Supporting symbols are $\dot{\forall}$, $\dot{\exists}$, $\dot{\natural}$, $\wedge_p$, and $\wedge_n$. The inference rules for these symbols are the same as the original symbols, $\forall$, $\exists$, and $\natural$. $\forall$ corresponds to $\natural$, and $\dot{\forall}$ corresponds to $\dot{\forall}$, so the definitions, which were defined as functions earlier, must be extended according to these correspondences. For example the definition of function $\varrho(\Phi, F, G, H)$ is extended as follows.

1. If no I-formulas of the form $\forall x I(x)$ or $\dot{\forall} x I(x)$ occur in H's input part, then,

$$\varrho(\Phi, F, G, H) \stackrel{def}{\equiv} H.$$

2. If some I-formulas of the form $\forall x I(x)$ or $\dot{\forall} x I(x)$ occur in H's input part, and O is the left-outermost I-formula of these I-formulas,

(a) If O has the form $\forall x.I(x)$, and J is the result of replacing O with $\natural x(\Phi_{(n)}, F, G, I(x))$, then,

$$\varrho(\Phi, F, G, H) \stackrel{def}{\equiv} J.$$

(b) If O has the form $\dot{\forall} x.I(x)$, and J is the result of replacing O with $\dot{\natural} x(\Phi_{(n)}, F, G, I(x))$, then,

$$\varrho(\Phi, F, G, H) \stackrel{def}{\equiv} J.$$

Definition 5.1 (Realizability Interpretation)
Let $\mathcal{M}$ be a structure, and F be an arbitrary formula that has no free variables, and ϵ be a Program Term. We define a binary relation (ϵ **realize** F) as follows.

1. ϵ **realize** $p_{(n)}a_1...a_n \stackrel{def}{\Longleftrightarrow} \epsilon\epsilon Set(p_{(n)}a_1...a_n) \wedge TDC'(\epsilon)$
2. ϵ **realize** $F \wedge_s G \stackrel{def}{\Longleftrightarrow} \exists e_1, e_2(\epsilon = (seq, e_1, e_2) \wedge (e_1$ **realize** $F) \wedge (e_2$ **realize** $G)) \wedge TDC'(\epsilon)$
3. ϵ **realize** $F \wedge G \stackrel{def}{\Longleftrightarrow} \exists e_1, e_2(\epsilon = (para, e_1, e_2) \wedge (e_1$ **realize** $F) \wedge (e_2$ **realize** $G)) \wedge TDC'(\epsilon)$
4. ϵ **realize** $F \wedge_n G \stackrel{def}{\Longleftrightarrow} \exists e_1, e_2(\epsilon = (nondet, e_1, e_2) \wedge (e_1$ **realize** $F) \wedge (e_2$ **realize** $G)) \wedge TDC'(\epsilon)$
5. ϵ **realize** $F \vee G \stackrel{def}{\Longleftrightarrow}$
   $\exists A, e_1, e_2(\epsilon = (A, e_1, e_2) \wedge ((A = l \wedge (e_1$ **realize** $F)) \vee (A = r \wedge (e_2$ **realize** $G)))) \wedge TDC'(\epsilon)$
6. ϵ **realize** $F \Rightarrow G \stackrel{def}{\Longleftrightarrow} \exists e_1(\epsilon = (\Pi, e_1) \wedge$
   $\forall A\,((A$ **realize** $F) \Rightarrow (e_1 A$ **realize** $G))) \wedge TDC'(\epsilon)$
7. ϵ **realize** $\neg F \stackrel{def}{\Longleftrightarrow} \forall A\,((A$ **realize** $F) \Rightarrow (\epsilon A$ **realize** $1)) \wedge TDC'(\epsilon)$
8. ϵ **realize** $\forall x F(x) \stackrel{def}{\Longleftrightarrow} \exists e_1(\epsilon = (in, e_1) \wedge \forall A\,(e_1 A$ **realize** $F(x))) \wedge TDC'(\epsilon)$
9. ϵ **realize** $\dot{\forall} x F(x) \stackrel{def}{\Longleftrightarrow} \exists e_1(\epsilon = (nonin, e_1) \wedge \forall A\,(e_1 A$ **realize** $F(x))) \wedge TDC'(\epsilon)$
10. ϵ **realize** $\exists x F(x) \stackrel{def}{\Longleftrightarrow} \exists A, e_1(\epsilon = (out, A, e_1) \wedge (e_1$ **realize** $F(A))) \wedge TDC'(\epsilon)$
11. ϵ **realize** $\dot{\exists} x.F(x) \stackrel{def}{\Longleftrightarrow} \exists A, e_1(\epsilon = (nonout, A, e_1) \wedge (e_1$ **realize** $F(A))) \wedge TDC'(\epsilon)$
12. ϵ **realize** $Ia_1...a_n \stackrel{def}{\Longleftrightarrow} Fix_I(\epsilon, a_1, ..., a_n)$
    where $Fix_I(Z, y_1, ..., y_n)$ is the least fixed point of the following relation.
    $Fix_I(Z, y_1, ..., y_n) \Leftrightarrow (TDC(Z) \wedge$
    (There exists some set S such that $< y_1, ..., y_n > \epsilon S$, and some wide input context $L(x_1, ..., x_n, \Phi)$, and some Program Terms $A(x_1, ..., x_n, y)$ and ϵ', such that

$$Z = \epsilon' y_1...y_n \wedge \forall x_1, ..., x_n([I/*][I/\Phi]\vartheta(L(x_1, ..., x_n, \Phi))) \equiv G(x_1, ..., x_n, I)$$
$$\wedge \epsilon' x_1...x_n = A(x_1, ..., x_n, \epsilon'))\wedge$$
$$\forall < x_1, ..., x_n > \epsilon S(A(x_1, ..., x_n, Z_\Phi)\ \mathbf{realize}_{<S, R_\Phi, Fix_I>}\ [R_\Phi/*]L(x_1, ..., x_n, \Phi))$$

    ))
    Where R_Φ is a fresh predicate variable.
    and the 5-ary relation (ϵ **realize**$_{<S,R,FIX>}$ F) is defined as follows.

    We abbreviate $< S, R, FIX >$ as κ.

(a) ϵ **realize**$_\kappa$ $\Psi_{(n)}a_1...a_n \stackrel{def}{\Longleftrightarrow} \epsilon = Z_{\Psi_{(n)}}a_1...a_n \wedge \mathbf{r}(\epsilon, S, \Psi_{(n)}a_1...a_n)$

(b) ϵ **realize**$_\kappa$ $p_{(n)}a_1...a_n \stackrel{def}{\Longleftrightarrow} \epsilon\epsilon Set(p_{(n)}a_1...a_n) \wedge \mathbf{r}(\epsilon, S, p_{(n)}a_1...a_n)$

(c) e **realize**$_\kappa$ $F \wedge_s G \stackrel{def}{\Longleftrightarrow}$
$\exists e_1, e_2 (e = (seq, e_1, e_2) \wedge (e_1$ **realize**$_\kappa$ $F) \wedge (e_2$ **realize**$_\kappa$ $G)) \wedge \mathbf{r}(e, S, F \wedge_s G)$

(d) e **realize**$_\kappa$ $F \wedge G \stackrel{def}{\Longleftrightarrow} \exists e_1, e_2 (e = (para, e_1, e_2) \wedge (e_1$ **realize**$_\kappa$ $F) \wedge (e_2$ **realize**$_\kappa$ $G)) \wedge \mathbf{r}(e, S, F \wedge G)$

(e) e **realize**$_\kappa$ $F \wedge_n G \stackrel{def}{\Longleftrightarrow}$
$\exists e_1, e_2 (e = (nondet, e_1, e_2) \wedge (e_1$ **realize**$_\kappa$ $F) \wedge (e_2$ **realize**$_\kappa$ $G)) \wedge \mathbf{r}(e, S, F \wedge_n G)$

(f) e **realize**$_\kappa$ $F \vee G \stackrel{def}{\Longleftrightarrow}$
$\exists A, e_1, e_2 (e = (A, e_1, e_2) \wedge ((A = l \wedge (e_1$ **realize**$_\kappa$ $F)) \vee (A = r \wedge (e_2$ **realize**$_\kappa$ $G)))) \wedge \mathbf{r}(e, S, F \vee G)$

(g) e **realize**$_\kappa$ $F \Rightarrow G \stackrel{def}{\Longleftrightarrow}$
$\exists e_1 (e = (\Pi, e_1) \wedge \forall A ((A$ **realize**$_\kappa$ $F) \Rightarrow (e_1 A$ **realize**$_\kappa$ $G))) \wedge \mathbf{r}(e, S, F \Rightarrow G)$

(h) e **realize** $\neg F \stackrel{def}{\Longleftrightarrow} \forall A ((A$ **realize**$_\kappa$ $F) \Rightarrow (e\, A$ **realize**$_\kappa$ $\bot)) \wedge \mathbf{r}(e, S, \neg F)$

(i) e **realize**$_\kappa$ $\forall x F(x) \stackrel{def}{\Longleftrightarrow} \exists e_1 (e = (in, e_1) \wedge \forall A (e_1 A$ **realize**$_\kappa$ $F(x))) \wedge \mathbf{r}(e, S, \forall x F(x))$

(j) e **realize**$_\kappa$ $\dot{\forall} x F(x) \stackrel{def}{\Longleftrightarrow} \exists e_1 (e = (nonin, e_1) \wedge \forall A (e_1 A$ **realize**$_\kappa$ $F(x))) \wedge \mathbf{r}(e, S, \dot{\forall} x F(x))$

(k) e **realize**$_\kappa$ $\exists x F(x) \stackrel{def}{\Longleftrightarrow} \exists A, e_1 (e = (out, A, e_1) \wedge (e_1$ **realize**$_\kappa$ $F(A))) \wedge \mathbf{r}(e, S, \exists x F(x))$

(l) e **realize**$_\kappa$ $\dot{\exists} x . F(x) \stackrel{def}{\Longleftrightarrow} \exists A, e_1 (e = (nonout, A, e_1) \wedge (e_1$ **realize**$_\kappa$ $F(A))) \wedge \mathbf{r}(e, S, \dot{\exists} x . F(x))$

(m) e **realize**$_\kappa$ $\natural x(\Phi, \lambda_{x_1 \ldots x_n} . H, I, G(x)) \stackrel{def}{\Longleftrightarrow} \exists A, e_1, e_2 (e = (in, \lambda_x . (if(x = A), e_1, e_2)) \wedge$
$(e_1$ **realize**$_\kappa$ $(I/\Phi_{(n)})G(A)) \wedge (e_2$ **realize**$_\kappa$ $G(x))) \wedge \mathbf{r}(e, S, \natural x(\Phi, \lambda_{x_1 \ldots x_n} . H, I, G(x)))$

(n) e **realize**$_\kappa$ $\dot{\natural} x(\Phi, \lambda_{x_1 \ldots x_n} . H, I, G(x)) \stackrel{def}{\Longleftrightarrow} \exists A, e_1, e_2 (e = (nonin, \lambda_x . (if(x = A), e_1, e_2))$
$\wedge (e_1$ **realize**$_\kappa$ $(I/\Phi_{(n)})G(A)) \wedge (e_2$ **realize**$_\kappa$ $G(x))) \wedge \mathbf{r}(e, S, \dot{\natural} x(\Phi, \lambda_{x_1 \ldots x_n} . H, I, G(x)))$

(o) e **realize**$_\kappa$ $Ia_1 \ldots a_n \stackrel{def}{\Longleftrightarrow} (e$ **realize** $Ia_1 \ldots a_n)$

(p) e **realize**$_{<S,R,FIX>}$ $Ra_1 \ldots a_n \stackrel{def}{\Longleftrightarrow} FIX(e, a_1, \ldots, a_n)$

Where,

- The meaning of the least fix point of the relation $R_n(x_1, \ldots x_n) \Leftrightarrow F(x_1, \ldots, x_n, R_n)$ ....(*) is the $n-ary$ relation γ which is the minimum relation in the set of $n-ary$ relations that satisfy (*), on the following partial order.

$$\gamma_n < \delta_n \stackrel{def}{\Longleftrightarrow} \{< x_1, \ldots, x_n > | \gamma_n(x_1, \ldots, x_n) holds\} \subset \{< x_1, \ldots, x_n > | \delta_n(x_1, \ldots, x_n) holds\}$$

- If $p_{(n)} a_1 \ldots a_n \epsilon Ax(\mathcal{M})$ then $Set(p_{(n)} a_1 \ldots a_n) = \{nop\}$ else $Set(p_{(n)} a_1 \ldots a_n) = \phi$.
- $I \equiv \mu . \lambda_{\Psi_{(n)} x_1 \ldots x_n} . G(x_1, \ldots, x_n, \Psi_{(n)})$.
- nop is the function from ϕ to ϕ.
- $\mathbf{r}(e, S, F) \stackrel{def}{\Longleftrightarrow} \mathbf{r}_1(e, F) \wedge \mathbf{r}_2(e, S, F)$, where $\mathbf{r}_1(e, F)$, $\mathbf{r}_2(e, S, F)$ is as follows.

  $\mathbf{r}_1(e, F)$

  If F has no occurrences of predicate free variables then $TDC(e)$ holds.

  $\mathbf{r}_2(e, S, F)$

  If F has some occurrences of predicate free variables, let e'' be the result of replacing all Program Term variables with e', where e' satisfies $\forall < b_1, \ldots, b_n >_{eS} (TIC(e' b_1 \ldots b_n))$, then,

$$tic(e'') \geq min_{<b_1, \ldots, b_n>_{eS}} (tic(e' b_1 \ldots b_n)).$$

  Especially if F has no predicate free variables except inside of $\natural$ or $\dot{\natural}$, then,

$$tic(e'') > min_{<b_1, \ldots, b_n>_{eS}} (tic(e' b_1 \ldots b_n)).$$

We have to show the existence of Fix_I in 12 as follows.

Claim

Let $I \equiv \mu.\lambda_{\Phi_{(n)},x_1,...,x_n}.G(x_1,...,x_n,\Psi_{(n)})$ and $F \equiv Ia_1...a_n$. Then the least fix point of the following relation exists.

$$Re_{n+1}(Z,y_1,...,y_n) \Leftrightarrow (TDC(Z) \wedge \exists S,L,A,\epsilon'(< y_1,...,y_n > \epsilon S \wedge Z = \epsilon' y_1...y_n \wedge$$
$$\forall x_1,...,x_n([I/*]([I/\Phi]\vartheta(L(x_1,...,x_n,\Phi))) \equiv G(x_1,...,x_n,I) \wedge \epsilon' x_1...x_n = A(x_1,...,x_n,\epsilon')) \wedge$$
$$\forall < x_1,...,x_n > \epsilon S(A(x_1,...,x_n,Z_\Phi)\ \mathbf{realize}_{<S,R_\Phi,Re_{n+1}>}\ [R_\Phi/*](L(x_1,...,x_n,\Phi)))))$$

Proof. First we define the function # as follows.

$$\#(G) \stackrel{def}{\equiv} \text{the number of the form } \mu.\lambda_{\Psi_{(n)},x_1,...,x_n}.H(x_1,...,x_n,\Psi_{(n)}) \text{ in } G$$

We show the existence of Re_{n+1}, by induction on $\#(G)$.

Base Case:

There is no μ in $G(x_1,...,x_n,\Psi_{(n)})$, because $\#(G) = 0$.

As the possible forms of L satisfying

$$\forall x_1,...,x_n([I/*]([I/\Phi]\vartheta(L(x_1,...,x_n,\Phi))) \equiv G(x_1,...,x_n,I)),$$

are finite, let $L_1,.....L_m$ be such forms, then the formula

$$(TDC(Z) \wedge \exists S,L,A,\epsilon'(< y_1,...,y_n > \epsilon S \wedge Z = \epsilon' y_1...y_n \wedge$$
$$\forall x_1,...,x_n([I/*]([I/\Phi]\vartheta(L(x_1,...,x_n,\Phi))) \equiv G(x_1,...,x_n,I) \wedge \epsilon' x_1...x_n = A(x_1,...,x_n,\epsilon')) \wedge$$
$$\forall < x_1,...,x_n > \epsilon S(A(x_1,...,x_n,Z_\Phi)\ \mathbf{realize}_{<S,R_\Phi,Re_{n+1}>}\ [R_\Phi/*](L(x_1,...,x_n,\Phi)))))$$

can be rewrite to

$$(TDC(Z) \wedge \exists S,A,\epsilon'(< y_1,...,y_n > \epsilon S \wedge Z = \epsilon' y_1...y_n \wedge$$
$$\forall x_1,....x_n(\epsilon' x_1...x_n = A(x_1,...,x_n,\epsilon')) \wedge$$
$$\bigvee_{0<i<n+1}(\forall < x_1,...,x_n > \epsilon S(A(x_1,...,x_n,Z_\Phi)\ \mathbf{realize}_{<S,R_\Phi,Re_{n+1}>}$$
$$[R_\Phi/*]L_i(x_1,...,x_n,\Phi))))).....(1)$$

Note that μ does not occur in L, because of the defintion of ϑ and the induction hypothesis.

By replacing all occurrences M of the form $(s\ \mathbf{realize}_{<S,R_\Phi,Re_{n+1}>}\ J)$ in formula (1) with the definition of $\mathbf{realize}_{<S,R_\Phi,Re_{n+1}>}$, repeatedly, we can get a formula K where all primitive relations are "TDC", "$=$", "$\equiv$", "ϵ", "$\mathbf{r}$", or "Re_{n+1}".

Note that, except for Re_{n+1}, all primitive relations that occur in K are well-defined.

By construction of K, it suffices to show the existence of the least fixed point of

$$Re_{n+1}(\epsilon,y_1,...,y_n) \Leftrightarrow K.....(2)$$

We can regard "relation" as "set", because we can define isomorphism function β $\beta(r) = \{< a_1,...,a_n > | r(a_1,...,a_n)\ \text{hold}\}$. Hence, we can regard K as the function in the following set.

$$\{S | S = \{< x_1,...x_{n+1} > | r(x_1,...,x_{n+1})\}, r \text{ is (n+1)-ary relation}\}$$

Therefore, it suffices to show that K is monotone as such a function.

$*$ occurs only in the input part of $L_i(x_1,...,x_n,\Phi)$ $(0 < i < m+1)$. Hence, by definition $\mathbf{realize}$ $< S, R_\Phi, Re_{n+1} >$, R_{n+1} occurs only in the input part of K.

By structure induction on K, we can prove that if R_{n+1} occurs only in the input part of K, K is monotone.

Induction Case:

Similarly for **Base Case**. □

When (ϵ **realize** F) holds, we say that the Program Term ϵ is a realizer for the formula F.

Definition 5.2 (Relation Real)

We define 3-ary relation $\mathbf{Real}_{<S>}$ as follows.

We abbreviate $< S >$ as κ.

1. ϵ **Real**$_{<S>}$ $\Psi_{(n)}a_1...a_n \stackrel{def}{\Longleftrightarrow} \epsilon = Z_{\Phi_{(n)}}a_1...a_n \wedge \mathbf{r}(\epsilon, S, \Psi_{(n)}a_1...a_n)$

2. ϵ **Real**$_\kappa$ $p_{(n)}a_1...a_n \stackrel{def}{\Longleftrightarrow} \epsilon\epsilon Set(p_{(n)}a_1...a_n) \wedge \mathbf{r}(\epsilon, S, p_{(n)}a_1...a_n)$

3. ϵ **Real**$_\kappa$ $F \wedge_s G \stackrel{def}{\Longleftrightarrow} \exists e_1, e_2(\epsilon = (scq, e_1, e_2) \wedge (e_1$ **Real**$_\kappa$ $F) \wedge (e_2$ **Real**$_\kappa$ $G)) \wedge \mathbf{r}(\epsilon, S, F \wedge_s G)$

4. ϵ **Real**$_\kappa$ $F \wedge G \stackrel{def}{\Longleftrightarrow} \exists e_1, e_2(\epsilon = (para, e_1, e_2) \wedge (e_1$ **Real**$_\kappa$ $F) \wedge (e_2$ **Real**$_\kappa$ $G)) \wedge \mathbf{r}(\epsilon, S, F \wedge G)$

5. ϵ **Real**$_\kappa$ $F \wedge_n G \stackrel{def}{\Longleftrightarrow} \exists e_1, e_2(\epsilon = (nondet, e_1, e_2) \wedge (e_1$ **Real**$_\kappa$ $F) \wedge (e_2$ **Real**$_\kappa$ $G)) \wedge \mathbf{r}(\epsilon, S, F \wedge_n G)$

6. ϵ **Real**$_\kappa$ $F \vee G \stackrel{def}{\Longleftrightarrow}$
$\exists A, e_1, e_2(\epsilon = (A, e_1, e_2) \wedge ((A = l \wedge (e_1$ **Real**$_\kappa$ $F)) \vee (A = r \wedge (e_2$ **Real**$_\kappa$ $G)))) \wedge \mathbf{r}(\epsilon, S, F \vee G)$

7. ϵ **Real**$_\kappa$ $F \Rightarrow G \stackrel{def}{\Longleftrightarrow} \exists e_1(\epsilon = (\Pi, e_1) \wedge \forall A\,((A$ **Real**$_\kappa$ $F) \Rightarrow (e_1 A$ **Real**$_\kappa$ $G))) \wedge \mathbf{r}(\epsilon, S)$

8. ϵ **realize** $\neg F \stackrel{def}{\Longleftrightarrow} \forall A\,((A$ **Real**$_\kappa$ $F) \Rightarrow (\epsilon A$ **Real**$_\kappa$ $\bot)) \wedge \mathbf{r}(\epsilon, S, F \Rightarrow G)$

9. ϵ **Real**$_\kappa$ $\forall x F(x) \stackrel{def}{\Longleftrightarrow} \exists e_1(\epsilon = (in, e_1) \wedge \forall A\,(e_1 A$ **Real**$_\kappa$ $F(x))) \wedge \mathbf{r}(\epsilon, S, \forall x F(x))$

10. ϵ **Real**$_\kappa$ $\dot{\forall} x F(x) \stackrel{def}{\Longleftrightarrow} \exists e_1(\epsilon = (nonin, e_1) \wedge \forall A\,(e_1 A$ **Real**$_\kappa$ $F(x))) \wedge \mathbf{r}(\epsilon, S, \dot{\forall} x F(x))$

11. ϵ **Real**$_\kappa$ $\exists x F(x) \stackrel{def}{\Longleftrightarrow} \exists A, e_1(\epsilon = (out, A, e_1) \wedge (e_1$ **Real**$_\kappa$ $F(A))) \wedge \mathbf{r}(\epsilon, S, \exists x F(x))$

12. ϵ **Real**$_\kappa$ $\dot{\exists} x F(x) \stackrel{def}{\Longleftrightarrow} \exists A, e_1(\epsilon = (nonout, A, e_1) \wedge (e_1$ **Real**$_\kappa$ $F(A))) \wedge \mathbf{r}(\epsilon, S, \dot{\exists} x F(x))$

13. ϵ **Real**$_\kappa$ $\natural x(\Phi, \lambda_{x_1,\ldots,x_n} H, I, G(x)) \stackrel{def}{\Longleftrightarrow} \exists A, e_1, e_2(\epsilon = (in, \lambda_x.(if(x = A), e_1, e_2)) \wedge$
$(e_1$ **Real**$_\kappa$ $\langle I/\Phi_{(n)}\rangle G(A)) \wedge (e_2$ **Real**$_\kappa$ $G(x))) \wedge \mathbf{r}(\epsilon, S, \natural x(\Phi, \lambda_{x_1,\ldots,x_n} H, I, G(x)))$

14. ϵ **Real**$_\kappa$ $\dot{\natural} x(\Phi, \lambda_{x_1,\ldots,x_n} H, I, G(x)) \stackrel{def}{\Longleftrightarrow} \exists A, e_1, e_2(\epsilon = (nonin, \lambda_x.(if(x = A), e_1, e_2))$
$\wedge (e_1$ **Real**$_\kappa$ $\langle I/\Phi_{(n)}\rangle G(A)) \wedge (e_2$ **Real**$_\kappa$ $G(x))) \wedge \mathbf{r}(\epsilon, S, \dot{\natural} x(\Phi, \lambda_{x_1,\ldots,x_n} H, I, G(x)))$

15. ϵ **Real**$_\kappa$ $Ia_1...a_n \stackrel{def}{\Longleftrightarrow} (\epsilon$ **realize** $Ia_1...a_n)$

where $I \equiv \mu.\lambda_{\Psi_{(n)},x_1,\ldots,x_n} G(x_1, ..., x_n, \Psi_{(n)})$, and the definiton of $\mathbf{r}$ is the same as in **definition** 5.1.

Lemma 5.1

ϵ **realize** $Ia_1...a_n \Leftrightarrow (TDC'(\epsilon) \wedge$
(There exists some set S such that $< a_1, ... a_n > \epsilon S$, and some wide input context $L(x_1, ..., x_n, \Phi)$, and some Program Terms $A(x_1, ..., x_n, y)$ and ϵ', such that,

$$\epsilon = \epsilon' a_1...a_n \wedge \forall x_1, ..., x_n([I/*]([I/\Phi]\vartheta(L(x_1, ..., x_n, \Phi))) \equiv G(x_1, ..., x_n, I)$$
$$\wedge \epsilon' x_1...x_n = A(x_1, ..., x_n, \epsilon')) \wedge$$
$$\forall < x_1, ..., x_n > \epsilon S(A(x_1, ..., x_n, Z_\Phi)\ \mathbf{Real}_{<S>}\ [I/*]L(x_1, ... x_n, \Phi))$$

))

Proof. This follows immediately from 12 in **definition** 5.1. □

Theorem 5.1

Let F be an arbitrary I-formula. If there exists an I-formula e, such that (e **Real**$_\kappa$ $\neg F$*) or (e* **realize** $\neg F$*) holds, then $e = nop$.*

Proof. Asume that r is a Program Term where $\epsilon = r$ and $r \neq nop$ stand. By the definition of realizability interpretation, there exists some set S_F of Program Terms, such that $\phi \neq S_F = \{A|(A$ **Real**$_\kappa$ $F)\}$ or $\phi \neq S_F = \{A|(A$ **realize** $F)\}$, and $e'\epsilon S_F \Rightarrow \epsilon e'\epsilon\phi$. This creates a contradiction.

□

Theorem 5.2
Let F be an arbitrary I-formula. if some formulas of the form $\natural x(\Phi_{(n)}, \lambda_{x_1,...,x_n}.H, I, G(x))$ or the form $\flat x(\Phi_{(n)}, \lambda_{x_1,...,x_n}.H, I, G(x))$ occur in F, then

$$((\epsilon \ \mathbf{Real}_\kappa \ F) \wedge \forall x_1, ..., x_n(< x_1, ..., x_n > \epsilon S \Rightarrow (\epsilon' x_1 ... x_n \ \mathbf{realize} \ I x_1 ... x_n))) \\ \Rightarrow ([\epsilon'/Z_{\Phi_{(n)}}]\epsilon \ \mathbf{realize} \ [I/\Phi_{(n)}]\partial(F))$$

stands for arbitrary Program Terms ϵ, ϵ' and for arbitrary set S.

Proof. By formula induction on F. □

Remark 5.1
Let F be I-formula. If no predicate free variables occur in F, then

$$(\epsilon \ \mathbf{Real}_\kappa \ F) \Leftrightarrow (\epsilon \ \mathbf{realize} \ F)$$

5.2 Program Extraction Rules

If $F_1, ..., F_n \vdash F$ is a provable sequent, and Program Terms that realize $F_1, ..., F_n$ are given, then we can extract the Program Term that realizes F. Therefore, if $\vdash F$ is a provable sequent, then we can get the Program Term that realizes F. In this subsection, we introduce the program extraction rules by which we extract the program from the proof.

Definition 5.3 (Realizing Variable, etc)
Let $F_1, ..., F_n \vdash F$ be a sequent. Then we call the Program Term Variable which corresponds to realizers for F_i, the "realizing variable". We write it $Rv(F_i)$, and the realizer of F, $Ext(F)$.

Definition 5.4 (Realizability of the Sequent)
In logic μ, we define the realizer for the sequent $F_1, ..., F_n \vdash F$ as follows.

- ϵ **realize** $F_1, ..., F_n \vdash F$
  $\stackrel{def}{\Longleftrightarrow} \forall x_1, ..., x_m, \epsilon_1, ..., \epsilon_n(R(n, F_i, \epsilon_i) \Rightarrow (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.\epsilon)\epsilon_n...)\epsilon_2)\epsilon_1 \ \mathbf{realize} \ F)$
- ϵ $\mathbf{Real}_\kappa$ $F_1, ..., F_n \vdash F \stackrel{def}{\Longleftrightarrow}$
  $\forall x_1, ..., x_m, \epsilon_1, ..., \epsilon_n(R_\kappa(n, F_i, \epsilon_i) \Rightarrow (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.\epsilon)\epsilon_n...)\epsilon_2)\epsilon_1 \ \mathbf{Real}_\kappa \ F)$.

where $x_1, ..., x_m$ are object free variables in $F_1, ..., F_n \vdash F$, $\epsilon_1, ..., \epsilon_n$ are Program Term variables, and

$$R(n, F_i, \epsilon_i) \stackrel{def}{\Longleftrightarrow} (\epsilon_1 \ \mathbf{realize} \ F_1) \wedge \\ ((\lambda_{Rv(F_1)}.\epsilon_2)\epsilon_1 \ \mathbf{realize} \ F_2) \wedge ... \wedge ((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_{n-1})}.\epsilon_n)\epsilon_{n-1}...)\epsilon_2)\epsilon_1 \ \mathbf{realize} \ F_n)$$
$$R_\kappa(n, F_i, \epsilon_i) \stackrel{def}{\Longleftrightarrow} (\epsilon_1 \ \mathbf{Real}_\kappa \ F_1) \wedge ((\lambda_{Rv(F_1)}.\epsilon_2)\epsilon_1 \ \mathbf{Real}_\kappa \ F_2) \wedge \\ ... \wedge ((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_{n-1})}.\epsilon_n)\epsilon_{n-1}...)\epsilon_2)\epsilon_1 \ \mathbf{Real}_\kappa \ F_n).$$

Definition 5.5 (Program Extraction Rules)
Let Ax be an axiom. We inductively define the realizer extraction rules which correspond to each inference rule, as follows.

- $Ext\left(\dfrac{}{\Gamma \vdash F}\right) \stackrel{def}{\equiv} Rv(F)$ (where F is in Γ).

- $Ext\left(\dfrac{\Gamma \vdash F \quad \Gamma \vdash \overset{\cdot}{G}}{\Gamma \vdash F <\wedge> G}\right) \stackrel{def}{\equiv} (<seq>, Ext(\Gamma \vdash F), Ext(\Gamma \vdash G))$
  (where if $<\wedge>$ is $\wedge$ then $<seq>$ is *para*, if $<\wedge>$ is $\wedge_s$ then $<seq>$ is *seq*, and if $<\wedge>$ is $\wedge_n$ then $<seq>$ is *nondet*.)

- $Ext\left(\dfrac{\Gamma \vdash F \wedge G}{\Gamma \vdash F}\right) \stackrel{def}{\equiv} proj(2)Ext(\Gamma \vdash F \wedge G)$
- $Ext\left(\dfrac{\Gamma \vdash F \wedge G}{\Gamma \vdash G}\right) \stackrel{def}{\equiv} proj(3)Ext(\Gamma \vdash F \wedge G)$
- $Ext\left(\dfrac{\Gamma \vdash F}{\Gamma \vdash F \vee G}\right) \stackrel{def}{\equiv} (l, (Ext(\Gamma \vdash F), nop))$
- $Ext\left(\dfrac{\Gamma \vdash G}{\Gamma \vdash F \vee G}\right) \stackrel{def}{\equiv} (r, (nop, Ext(\Gamma \vdash F)))$
- $Ext\left(\dfrac{\Gamma \vdash F \vee G \quad \Gamma, F \vdash H \quad \Gamma, G \vdash H}{\Gamma \vdash H}\right)$
  $\stackrel{def}{\equiv} select(proj(1)Ext(\Gamma \vdash F \vee G), (\lambda_{Rv(F)}.Ext(\Gamma, F \vdash H))$
  $(proj(1)(proj(2)Ext(\Gamma \vdash F \vee G))), (\lambda_{Rv(G)}.Ext(\Gamma, G \vdash H))(proj(2)(proj(2)Ext(\Gamma \vdash F \vee G))))$
- $Ext\left(\dfrac{\Gamma, F \vdash G}{\Gamma \vdash F \Rightarrow G}\right) \stackrel{def}{\equiv} (\Pi, \lambda_{Rv(F)}.Ext(\Gamma, F \vdash G))$
- $Ext\left(\dfrac{\Gamma \vdash F \Rightarrow G \quad \Gamma \vdash F}{\Gamma \vdash G}\right) \stackrel{def}{\equiv} (proj(2)Ext(\Gamma \vdash F \Rightarrow G))Ext(\Gamma \vdash F)$
- $Ext\left(\dfrac{\Gamma, y \downarrow \vdash F(y)}{\Gamma \vdash <\forall> xF(x)}\right) \stackrel{def}{\equiv} (<in>, \lambda_x.Ext(\Gamma \vdash F(x)))$
  (where if $<\forall>$ is $\forall$ then $<in>$ is *in*, and if $<\forall>$ is $\dot{\forall}$ then $<in>$ is *nonin*.)

- $Ext\left(\dfrac{\Gamma \vdash \forall xF(x) \quad \Gamma \vdash a \downarrow}{\Gamma \vdash F(a)}\right) \stackrel{def}{\equiv} (proj(2)Ext(\Gamma \vdash \forall xF(x)))a$
- $Ext\left(\dfrac{\Gamma, \vdash F(a) \quad \Gamma \vdash a \downarrow}{\Gamma \vdash <\exists> xF(x)}\right) \stackrel{def}{\equiv} (<out>, a, Ext(\Gamma \vdash F(a)))$
  (where if $<\exists>$ is $\exists$ then $<out>$ is *out*, and if $<\exists>$ is $\dot{\exists}$ then $<out>$ is *nonout*.)

- $Ext\left(\dfrac{\Gamma \vdash \exists xF(x) \quad \Gamma, F(a) \vdash G}{\Gamma \vdash G}\right)$
  $\stackrel{def}{\equiv} (\lambda_{Rv(F(a))}.(\lambda_a.Ext(\Gamma, F(a) \vdash G))(proj(2)Ext(\Gamma \vdash \exists xF(x))))(proj(3)Ext(\Gamma \vdash \exists xF(x)))$
- $Ext\left(\dfrac{\Gamma, F \vdash \bot}{\Gamma \vdash \neg F}\right) \stackrel{def}{\equiv} \lambda_{Rv(F)}.Ext(\Gamma, F \vdash \bot)$
- $Ext\left(\dfrac{\Gamma \vdash \bot}{\Gamma \vdash A}\right) \stackrel{def}{\equiv} nop$
- $Ext\left(\dfrac{\Gamma, \forall y_1, ..., y_n(<y_1, ..., y_n> <_{jn} <x_1, ..., x_n> \Rightarrow F(y_1, ..., y_n)) \vdash F(x_1, ..., x_n)}{\Gamma \vdash <\forall x_1, ..., x_n> F(x_1, ..., x_n)}\right)$
  $\stackrel{def}{\equiv} (<in>_1, \lambda_{x_1}.(...(<in>_n, \lambda_{x_n}.(\mu.(\lambda_Z.\lambda_{x_1,...,x_n}.((\lambda_{Rv(\xi)}.Ext(\zeta)))$
  $(in, \lambda_{y_1}.(...(in, \lambda_{y_n}.(\Pi, \lambda_x.Z y_1 ... y_n))...))))x_1 ... x_n)...))$
  (where $\xi \equiv \forall y_1, ..., y_n.((y_1, ..., y_n) <_{jn} (x_1, ..., x_n) \Rightarrow F(y_1, ..., y_n))$,
  $\zeta \equiv \Gamma, \forall y_1, ... y_n.((y_1, ..., y_n) <_{jn} (x_1, ..., x_n) \Rightarrow F(y_1, ..., y_n)) \vdash F(x_1, ..., x_n)$,
  $<\forall x_1, ..., x_n>$ show $<\forall>_1 x_1 ... <\forall>_n x_n$, if $<\forall>_t$ is $\dot{\forall}$ then $<in>_t$ is *nonin*, else $<in>_t$ is *in*.)

- $Ext\left(\dfrac{\Gamma \vdash G(b_1,...,b_n, Ia_{11}...a_{1n},...., Ia_{m1}...a_{mn})}{\Gamma \vdash Ib_1...b_n}\right)$

  $\stackrel{def}{\equiv} Ext(\Gamma \vdash G(b_1,...,b_n, Ia_{11}...a_{1n},..., Ia_{m1}...a_{mn}))$

  (where $I \equiv \mu.\lambda_{\Psi_{(n)}x_1....x_n}.G(x_1,....x_n, \Psi a_{11}...a_{1n},.... \Psi a_{m1}...a_{mn})$).

- $Ext\left(\dfrac{\Gamma \vdash Ib_1...b_n}{\Gamma \vdash G(b_1,...,b_n, Ia_{11}...a_{1n},.... Ia_{m1}...a_{mn})}\right)$

  $\stackrel{def}{\equiv} Ext(\xi)$

  (where $I \equiv \mu.\lambda_{\Psi_{(n)}x_1....x_n}.G(x_1,....x_n, \Psi a_{11}...a_{1n},.... \Psi a_{m1}...a_{mn})$, $\xi \equiv \Gamma \vdash Ib_1...b_n$).

- $Ext\left(\dfrac{\Gamma \vdash \exists x(I/\Phi_{(n)})G(x) \quad \Gamma, \forall x_1,....,x_n(H \Rightarrow \Phi_{(n)}x_1...x_n) \vdash \forall x G(x)}{\Gamma \vdash <\natural> x(\Phi, \lambda_{x_1....x_n}.H, I, G(x))}\right)$

  $\stackrel{def}{\equiv} (< in >, \lambda_x.select(if(x = (proj(2)Ext(\varsigma)), proj(3)Ext(\varsigma), (proj(2)((\lambda_{R\eta(\zeta)}.Ext(\xi))$

  $(in, \lambda_{x_1}.(...(in, \lambda_{x_n}.(\Pi, \lambda_y.Z_{\Phi_n}x_1...x_n))...)))x))$

  (where if $<\natural>$ is $<\hat{\natural}>$ then $< in >$ is *nonin*, if $<\natural>$ is $\natural$ then $< in >$ is *in*, $\varsigma \equiv \Gamma \vdash \exists x(I/\Phi)G(x)$,

  $\xi \equiv \Gamma, \forall x_1,...x_n(H(x_1,...,x_n) \Rightarrow \Phi_{(n)}x_1...x_n) \vdash \forall x G(x)$, $\zeta \equiv \forall x_1,...x_n(H \Rightarrow \Phi_{(n)}x_1...x_n)$,

  $I \equiv \mu.\lambda_{\Psi_{(n)}x_1....x_n}.G(x_1,...,x_n, \Psi)$).

- $Ext\left(\dfrac{\Gamma, H(y_1,...,y_n) \vdash \varpi(\Phi, \lambda_{x_1,...x_n}.H(x_1,...,x_n), I, K(y_1,...,y_n)) \quad \Gamma \vdash H(a_1,....a_n)}{\Gamma \vdash Ia_1...a_n}\right)$

  $\stackrel{def}{\equiv} (\mu.\lambda_{Z_{\Psi_{(n)}}}.\lambda_{y_1,...y_n}.(\lambda_{R\eta(\xi)}.Ext(\varsigma))nop)a_1...a_n$

  (where $I \equiv \mu.\lambda_{\Psi_{(n)}x_1....x_n}.G(x_1,...,x_n, \Psi a_{11}...a_{1n},.... \Psi a_{m1}...a_{mn})$,

  $K(y_1,...,y_n) \equiv G(y_1,...,y_n, \Phi a_{11}...a_{1n},...,\Phi a_{m1}...a_{mn})$,

  $\varsigma \equiv \Gamma, H(y_1,...,y_n) \vdash \varpi(\lambda_{x_1,...x_n}.H(x_1,...,x_n), I_{(n)}, K(y_1,...,y_n))$,

  $\xi \equiv H(y_1,...,y_n)$).

- $Ext\left(\dfrac{\Gamma \vdash a \downarrow}{\Gamma \vdash a = a}\right) \stackrel{def}{\equiv} nop$
- $Ext\left(\dfrac{\Gamma \vdash a = b \quad \Gamma \vdash F(a)}{\Gamma \vdash F(b)}\right) \stackrel{def}{=} Ext(\Gamma \vdash F(a))$
- $Ext\left(\dfrac{\Gamma \vdash a_1 < a_2 \quad \Gamma \vdash a_2 < a_3}{\Gamma \vdash a_1 < a_3}\right) \stackrel{def}{=} nop$
- $Ext\left(\dfrac{}{\Gamma \vdash A}\right) \stackrel{def}{\equiv} nop$ (where $AxAx$)

Remark 5.2

In $Ext(\Gamma \vdash F)$, there are no object free variables except the object free variables in $\Gamma \vdash F$.

Theorem 5.3 (Soundness)

If δ is a provable sequent of logic μ, then ($Ext(\delta)$ **realize** *δ).*

Proof. See **Appendix 1**. □

6 Example

We extracted the Program Term which is specified by the following formula. Roughly, the execution of this program is as shown,

"If you input the sequence 2,3,4,5,... then you will be given the output sequence 2,3,5,7, 11,13,17,.... If you want to halt the program, input 0".

Specification:

$$(\mu.\lambda_{\Psi:i\_list-prop}.\lambda_{p:i\_list}.\forall x : int(\neg x = 0 \wedge (\neg div\_list(x,p) \wedge \exists y : int(x = y \wedge_s \dot{\exists} k : i\_list(k = cons(x,p) \wedge \Psi k))) \vee (div\_list(x,p) \wedge \Psi p) \vee x = 0 \wedge \exists y : string(y =' OK')))nil,$$

where $div\_list(a,b)$ is a formula where if there exists an element of list b such that the element can be divided by a, then true. As we implement this logic to the system as the typed logic, the above specification is expressed by the typed formula. The proof of this formula, and the Program Term that is extracted from it, are shown in the appendixes A and B.

7 Acknowledgment

I would like to thank Ko Sakai for many hours of stimulating discussions on termination properties and especially for his help in formulating the concept of Program Term.

References

Howard,W.A.(1984). The formulae-as-types notion of construction. in *Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, 479-490.

Martin-Löf,P.(1982). Constructive mathematics and computer programming. in *Logic, Methodology, and Philosophy of Science VI*, Cohen, L.J. et al. eds., North-Holland, 153-179.

Sato,M.(1986). QJ: A Constructive Logical System with Types. France-Japan Artificial Intelligence and Computer Science Symposium 86, Tokyo.

Hayashi,S. and Nakano,H.(1987). PX: a computational logic. RIMS-573, RIMS, Kyoto University.

Takayama,Y.(1989). Extended Projection — a new technique to extract efficient programs from constructive proofs. Proceedings of 1989 Conference on Functional Programming Languages and Computer Architecture, ACM.

Milner,R.(1989). Communication and Concurrency. Prentice-Hall International.

Hoare,C.(1985). Communicating Sequential Processes. Prentice-Hall International.

Appendix 1
Soundness

Proof. By induction on proof $\Im$. It suffices to show

1. if the lowest inference rule of $\Im$ is

$$\frac{\delta_1 \quad \delta_2 \quad \ldots \delta_m}{\delta} \ \mathrm{R}$$

A is a Program Term that is extracted from $Ext(\delta_i)$ $(0 < i < m+1)$, and R is not Mu-Intro or Natural-Intro, then,

(a) $\forall i \epsilon \{1, 2, \ldots, m\}((Ext(\delta_i)\ \mathbf{realize}\ \delta_i)) \Rightarrow (A\ \mathbf{realize}\ \delta)$

(b) for all κ, $\forall i \epsilon \{1, 2, \ldots, m\}((Ext(\delta_i)\ \mathbf{Real}_\kappa\ \delta_i)) \Rightarrow (A\ \mathbf{Real}_\kappa\ \delta)$

2. If the lowest inference rule of $\Im$ is Natural-Intro:

$$\frac{\Gamma \vdash \exists x(I/\Phi)K(x) \quad \Gamma, \forall x_1, \ldots x_n(H \Rightarrow \Phi_{(n)} x_1 \ldots x_n) \vdash \forall x K(x)}{\Gamma \vdash \natural x(\Phi_{(n)}, \lambda_{x_1, \ldots x_n}.H, I, G(x))}$$

(Assume $I \equiv \mu.\lambda_{\Phi_{(n)} x_1, \ldots x_n}.G(x_1, \ldots, x_n, \Psi_{(n)})$.)

Let δ_1, δ_2 be the upper formulas of the inferece, δ be the lower fomula, A be the Program Term that is extracted from $Ext(\delta_i)$ $(0 < i < 3)$, and S be

$$\{< b_1, \ldots, b_n > \mid \vdash H(b_1, \ldots, b_n) \text{is a provable sequent of logic } \mu\},$$

then,

$$\forall i \epsilon \{1, 2\}((Ext(\delta_i)\ \mathbf{Real}_{<S>}\ \delta_i)) \Rightarrow (A\ \mathbf{Real}_{<S>}\ \delta)$$

3. If the lowest inference rule of $\Im$ is Mu-Intro:

$$\frac{\Gamma, H(y_1, \ldots, y_n) \vdash \varpi(\Phi_{(n)}, \lambda_{x_1, \ldots x_n}.H(x_1, \ldots, x_n), I, K(y_1, \ldots, y_n)) \quad \Gamma \vdash H(a_1, \ldots, a_n)}{\Gamma \vdash I a_1 \ldots a_n}$$

(Assume that $H(y_1, \ldots, y_n) \equiv \neg L(y_1, \ldots, y_n)$ for some L, $I \equiv \mu.\lambda_{\Phi_{(n)} x_1, \ldots x_n}.G(x_1, \ldots, x_n, \Psi)$, $K(y_1, \ldots, y_n) \equiv G(y_1, \ldots, y_n, I)$, and $\Phi_{(n)}$ does not appear in the lower formula.)

Let δ_1, δ_2 be the upper formulas of the inference, δ be the lower formula, A be the Program Term that is extracted from $Ext(\delta_i)$ $(0 < i < 3)$, and S be

$$\{< b_1, \ldots, b_n > \mid \vdash H(b_1, \ldots, b_n) \text{is the provable sequent on logic } \mu\}$$

then,

$$\forall i \epsilon \{1, 2\}((Ext(\delta_i)\ \mathbf{Real}_{<S>}\ \delta_i)) \Rightarrow (A\ \mathbf{realize}\ \delta)$$

The proofs of each case are as follows:

1. Each proof for (a) and (b) are shown as follows:

- Axiom, Assumption
  It follows immediately from the definition of the Program Extraction rule and the realizability interpretation.
- And-Intro
  - $\wedge_2$
    Let Γ be $F_1, \ldots, F_n$, and $x_1, \ldots, x_m$ be the object free variables in $\Gamma \vdash F$ and $\Gamma \vdash G$.

    (a) By induction hypothesis

$$\forall x_1, ..., x_m, e_1, ..., e_n (R(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.Ext(\Gamma \vdash F)e_n...)e_2)e_1 \textbf{ realize } F) \quad .....(1)$$

$$\forall x_1, ..., x_m, e_1, ..., e_n (R(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.Ext(\Gamma \vdash G)e_n...)e_2)e_1 \textbf{ realize } G) \quad .....(2)$$

By the definition of the program extraction rule

$$Ext(\Gamma \vdash F \wedge G) = (seq, Ext(\Gamma \vdash F), Ext(\Gamma \vdash G))$$

Note that we have:

$$\forall x_1, ..., x_m, e_1, ..., e_n (R(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.(seq, Ext(\Gamma \vdash F), Ext(\Gamma \vdash G)))e_n...)e_2)e_1$$
$$= (seq, (\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.Ext(\Gamma \vdash F)e_n...)e_2)e_1,$$
$$(\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.Ext(\Gamma \vdash G)e_n...)e_2)e_1)),$$

hence, by (1) and (2),

$$\exists A_1, A_2, \forall x_1, ..., x_m, e_1, ..., e_n (R(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.(seq, Ext(\Gamma \vdash F), Ext(\Gamma \vdash G)))e_n...)e_2)e_1$$
$$= (seq, A_1, A_2) \wedge (A_1 \textbf{ realize } F) \wedge (A_2 \textbf{ realize } G))$$

Let $x_1, ..., x_m$ be arbitrary objects that have values, and $e_1, ..., e_n$ be arbitrary Program Terms. By induction hypothesis

$$R(n, F_i, e_i) \Rightarrow TDC((...((\lambda_{Rn(F_1)...Rn(F_n)}.Ext(\Gamma \vdash F)e_n...)e_2)e_1) \wedge$$
$$TDC((\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.Ext(\Gamma \vdash G)e_n...)e_2)e_1)$$

So use **Lemma** 2.2,

$$R(n, F_i, e_i) \Rightarrow TDC((seq, (\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.Ext(\Gamma \vdash F)e_n...)e_2)e_1,$$
$$(\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.Ext(\Gamma \vdash G)e_n...)e_2)e_1))$$

(b) By induction hypothesis,

$$\forall x_1, ..., x_m, e_1, ..., e_n (R_n(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.Ext(\Gamma \vdash F)e_n...)e_2)e_1 \textbf{ Real}_n \ F) \quad .....(1)$$

$$\forall x_1, ..., x_m, e_1, ..., e_n (R_n(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.Ext(\Gamma \vdash G)e_n...)e_2)e_1 \textbf{ Real}_n \ G) \quad .....(2)$$

By the definition of the program extraction rule

$$Ext(\Gamma \vdash F \wedge G) = (seq, Ext(\Gamma \vdash F), Ext(\Gamma \vdash G))$$

Note that we have:

$$\forall x_1, ..., x_m, e_1, ..., e_n (R_n(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.(seq, Ext(\Gamma \vdash F), Ext(\Gamma \vdash G)))e_n...)e_2)e_1$$
$$= (seq, (\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.Ext(\Gamma \vdash F)e_n...)e_2)e_1,$$
$$(\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.Ext(\Gamma \vdash G)e_n...)e_2)e_1)),$$

hence, by (1) and (2),

$$\exists A_1, A_2, \forall x_1, ..., x_m, e_1, ..., e_n (R_n(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.(seq, Ext(\Gamma \vdash F), Ext(\Gamma \vdash G)))e_n...)e_2)e_1$$
$$= (seq, A_1, A_2) \wedge (A_1 \textbf{ Real}_n \ F) \wedge (A_2 \textbf{ Real}_n \ G)) \quad ....(3)$$

* $\mathbf{r}_1$

Case $F \wedge G$ contains no predicate free variables.

Clearly, F and G contain no prediate free variables either. Let $x_1, ..., x_m$ be arbitrary objects that have values, $e_1, ..., e_n$ be arbitrary Program Terms, and assume $R_n(n, F_i, e_i)$. By induction hypothesis,

$$TDC((\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.Ext(\Gamma \vdash F)e_n...)e_2)e_1),$$
$$TDC((\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.Ext(\Gamma \vdash G)e_n...)e_2)e_1),$$

hence, by **Lemma** 2.2

$$R_n(n, F_i, e_i) \Rightarrow TDC((seq, (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F))e_n...)e_2)e_1, \\ (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash G))e_n...)e_2)e_1))$$

- r_2

  Case $F \wedge G$ has some occurrence of predicate free variables .

  Let $x_1, ..., x_m$ be arbitrary objects that have values, $e_1, ..., e_n$ be arbitrary Program Terms, and e'_F, e'_G, and $e'_{F \wedge G}$ be Program Terms that are the results of replacing program variable $Z_{\Psi_{(j)}}$ with r in e_F, e_G, and $e_{F \wedge G}$ respectively, where

  $$e_F = (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F))e_n...)e_2)e_1$$
  $$e_G = (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash G))e_n...)e_2)e_1$$
  $$e_{F \wedge G} = (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(seq.\ Ext(\Gamma \vdash F), Ext(\Gamma \vdash G)))e_n...)e_2)e_1$$

  and ϵ satisfies

  $$\forall < b_1, ..., b_j >_{\epsilon S} (TIC'(eb_1...b_j)).$$

  By induction hypothesis

  $$tic(e'_F) \geq min_{<b_1,...,b_i>_{\epsilon S}}(tic(eb_1...b_j)), \qquad tic(e'_G) \geq min_{<b_1,...,b_j>_{\epsilon S}}(tic(eb_1...b_j))$$

  hence, by **Lemma** 2.2 and (3),

  $$tic(e'_{F \wedge G}) \geq min_{<b_1,...,b_j>_{\epsilon S}}(tic(eb_1...b_j)).$$

  Case $F \wedge G$ has no predicate free variables except for inside of $\natural$ or $\hat{\natural}$, the predicate free variables in F and G deservedly occur only in $\natural$ and $\hat{\natural}$, hence

  $$tic(e'_{F \wedge G}) > min_{<b_1,...,b_j>_{\epsilon S}}(tic(eb_1...b_j))$$

  can follow similarly.

◇ $\wedge$, $\wedge_n$

Similarly for $\wedge$.

- And-Elim 1

◇ $\wedge_e$

Let Γ be $F_1, ..., F_n$, $x_1, ..., x_m$ be object free variables in $\Gamma \vdash F \wedge G$, and $x_1, ..., x_j$ $(j \leq m)$ be object free variables in $\Gamma \vdash F$.

(a) By induction hypothesis

$$\forall x_1, ..., x_m, e_1, ..., e_n(R(n, F_i, e_i) \Rightarrow \\ (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F \wedge G))e_n...)e_2)e_1 \textbf{ realize } F \wedge G) \quad .....(1)$$

By the definition of the program extraction rule

$$Ext(\Gamma \vdash F) = proj(2)(Ext(\Gamma \vdash F))$$

Hence,

$$\forall x_1, ..., x_m, e_1, ..., e_n(R(n, F_i, e_i) \Rightarrow \\ (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.proj(2)(\Gamma \vdash F \wedge G))e_n...)e_2)e_1 \\ = proj(2)((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F \wedge G))e_n...)e_2)e_1))$$

$x_1, ..., x_j$ be the object free variables in F, hence, by **Remark** 5.2

$$\forall x_1, ..., x_j, e_1, ..., e_n(R(n, F_i, e_i) \Rightarrow \\ (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.proj(2)(\Gamma \vdash F \wedge G))e_n...)e_2)e_1 \\ = proj(2)((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F \wedge G))e_n...)e_2)e_1)),$$

hence, by (1) and the definition of realizability interpretation of $\wedge$,

$$\exists A, \forall x_1, ..., x_j, e_1, ..., e_n(R(n, F_i, e_i) \Rightarrow \\ (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.proj(2)(\Gamma \vdash F \wedge G))e_n...)e_2)e_1 = A \wedge (A \textbf{ realize } F))$$

(b) Similarly for (a).

◇ $\wedge$, $\wedge_n$

Similary for $\wedge$.

- And-Elim 2
  Similarly for And-Elim 1.
- Or-Intro 1
  Let Γ be $F_1, ..., F_n$, $x_1, ..., x_m$ be object free variables in $\Gamma \vdash F \vee G$, and $x_1, ..., x_j$ $(j \leq m)$ be object free variables in $\Gamma \vdash F$.

(a) By induction hypothesis

$$\forall x_1, ..., x_j, e_1, ..., e_n(R(n, F_i, e_i) \Rightarrow (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F))e_n...)e_2)e_1 \textbf{ realize } F) \quad .....(1)$$

By the definition of the program extraction rule

$$Ext(\Gamma \vdash F \vee G) = (l, (Ext(\Gamma \vdash F), nop))$$

Hence,

$$\forall x_1, ..., x_j, e_1, ..., e_n(R(n, F_i, e_i) \Rightarrow (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(l, (Ext(\Gamma \vdash F \vee G), nop)))e_n...)e_2)e_1 = (l, ((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F))e_n...)e_2)e_1, nop)))$$

Since (1) and the definition of realizability interpretation of $\vee$,

$$\exists A, \forall x_1, ..., x_j, e_1, ..., e_n(R(n, F_i, e_i) \Rightarrow (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(l, (Ext(\Gamma \vdash F), nop)))e_n...)e_2)e_1 = (l, (A, nop)) \wedge (A \textbf{ realize } F))$$

By **Remark** 5.2, $x_{j+1}, ..., x_m$ do not occur in

$$(l, ((\lambda_{Rv(F_1)...Rv(F_n)}.Ext(\Gamma \vdash F))e_1...e_n, nop))$$

So

$$\exists A, \forall x_1, ..., x_m, e_1, ..., e_n(R(n, F_i, e_i) \Rightarrow (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(l, (Ext(\Gamma \vdash F \wedge G), nop)))e_n...)e_2)e_1 = (l, (A, nop)) \wedge (A \textbf{ realize } F))$$

By induction hypothesis and $(l, (A, nop)) \xrightarrow{\tau} A$

$TDC((l, (A, nop)))$.

(b) By induction hypothesis

$$\forall x_1, ..., x_j, e_1, ..., e_n(R_\kappa(n, F_i, e_i) \Rightarrow (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F))e_n...)e_2)e_1 \textbf{ Real}_\kappa F) \quad .....(1)$$

By the definition of the program extraction rule

$$Ext(\Gamma \vdash F \vee G) = (l, (Ext(\Gamma \vdash F), nop))$$

Hence,

$$\forall x_1, ..., x_s, e_1, ..., e_n(R_\kappa(n, F_i, e_i) \Rightarrow (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(l, (Ext(\Gamma \vdash F \vee G), nop)))e_n...)e_2)e_1 = (l, ((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F))e_n...)e_2)e_1, nop)))$$

Since (1) and the definition of realizability interpretation of $\vee$,

$$\forall x_1, ..., x_j, e_1, ..., e_n(R_\kappa(n, F_i, e_i) \Rightarrow (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(l, (Ext(\Gamma \vdash F), nop)))e_n...)e_2)e_1 = (l, (A, nop)) \wedge (A \textbf{ Real}_\kappa F))$$

By **Remark** 5.2, $x_{j+1}, ..., x_m$ do not occur in

$$(l, ((\lambda_{Rv(F_1)...Rv(F_n)}.Ext(\Gamma \vdash F))e_1...e_n, nop))$$

so

$$\forall x_1, ..., x_m, e_1, ..., e_n(R_\kappa(n, F_i, e_i) \Rightarrow (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(l, (Ext(\Gamma \vdash F \wedge G), nop)))e_n...)e_2)e_1 = (l, (A, nop)) \wedge (A \textbf{ Real}_\kappa F))$$

$\mathbf{r}(\epsilon, S, Fix)$ follows immediately form induction hypothesis and $(l, (A, nop)) \xrightarrow{r} A$

- Or-Intro 2

  Similarly for Or-Intro 1.

- Or-Elim

  Let Γ be $F_1, ..., F_n$, $w_1, ..., w_l$ be object free variables in Γ, $x_1, ..., x_j$ be object free variables in F, and $z_1, ..., z_m$ be object free variables in H.

(a) By induction hypothesis

$$\forall w_1, ..., w_h, x_1, ..., x_j, y_1, ..., y_k, \epsilon_1, ..., \epsilon_n (R(n, F_i, \epsilon_i) \Rightarrow$$
$$(\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F \wedge G))\epsilon_n...)\epsilon_2)\epsilon_1 \textbf{ realize } F \wedge G) \quad ....(1)$$

$$\forall w_1, ..., w_h, x_1, ..., x_j, z_1, ..., z_m, \epsilon_1, ..., \epsilon_n (R(n, F_i, \epsilon_i) \wedge$$
$$((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.\epsilon_F)\epsilon_n...)\epsilon_2)\epsilon_1 \textbf{ realize } F) \Rightarrow$$
$$((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(\lambda_{Rv(F)}.Ext(\Gamma, G \vdash H))\epsilon_F)\epsilon_n...)\epsilon_2)\epsilon_1$$
$$\textbf{realize } H)) \quad .....(2)$$

$$\forall w_1, ..., w_h, x_1, ..., x_j, z_1, ..., z_m, \epsilon_1, ..., \epsilon_n (R(n, F_i, \epsilon_i) \wedge$$
$$((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.\epsilon_G)\epsilon_n...)\epsilon_2)\epsilon_1 \textbf{ realize } F) \Rightarrow$$
$$((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(\lambda_{Rv(G)}.Ext(\Gamma, G \vdash H))\epsilon_G)\epsilon_n...)\epsilon_2)\epsilon_1$$
$$\textbf{realize } H)) \quad ....(3)$$

By the definition of the program extraction rule

$$Ext(\Gamma \vdash H) = select(proj(1)Ext(\Gamma \vdash F \vee G),$$
$$(\lambda_{Rv(F)}.Ext(\Gamma, F \vdash H))(proj(1)(proj(2)Ext(\Gamma \vdash F \vee G))),$$
$$(\lambda_{Rv(G)}.Ext(\Gamma, G \vdash H))(proj(2)(proj(2)Ext(\Gamma \vdash F \vee G))))$$

Note that we have

$$\forall w_1, ..., w_h, x_j, ..., x_j, y_1, ..., y_k, z_1, ..., z_m, \epsilon_1, ..., \epsilon_n (R(n, F_i, \epsilon_i) \Rightarrow$$
$$(\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.select(proj(1)Ext(\Gamma \vdash F \vee G),$$
$$(\lambda_{Rv(F)}.Ext(\Gamma, F \vdash H))(proj(1)(proj(2)Ext(\Gamma \vdash F \vee G))),$$
$$(\lambda_{Rv(G)}.Ext(\Gamma, G \vdash H))(proj(2)(proj(2)Ext(\Gamma \vdash F \vee G)))))\epsilon_n...)\epsilon_2)\epsilon_1$$
$$= select(proj(1))((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F \vee G))\epsilon_1...)\epsilon_2)\epsilon_1),$$
$$((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(\lambda_{Rv(F)}.Ext(\Gamma, F \vdash H))$$
$$(proj(1)(proj(2)Ext(\Gamma \vdash F \vee G))))\epsilon_1...)\epsilon_2)\epsilon_1),$$
$$((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(\lambda_{Rv(G)}.Ext(\Gamma, G \vdash H))$$
$$(proj(2)(proj(2)(Ext(\Gamma \vdash F \vee G)))))\epsilon_1...)\epsilon_2)\epsilon_1)))$$

Hence, by (1) and the definition of realizability interpretation of $\vee$,

$$\forall w_1, ..., w_h, x_j, ..., x_j, y_1, ..., y_k, z_1, ..., z_m, \epsilon_1, ..., \epsilon_n (R(n, F_i, \epsilon_i) \Rightarrow$$
$$((Ext(\Gamma \vdash H) =$$
$$select(l, (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(\lambda_{Rv(F)}.Ext(\Gamma, F \vdash H))\epsilon')\epsilon_1...)\epsilon_2)\epsilon_1,$$
$$(\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(\lambda_{Rv(G)}.Ext(\Gamma, G \vdash H))nop)\epsilon_1...)\epsilon_2)\epsilon_1)$$
$$\wedge((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.\epsilon')\epsilon_1...)\epsilon_2)\epsilon_1 \textbf{ realize } F))$$
$$\vee(Ext(\Gamma \vdash H) =$$
$$select(r, (\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(\lambda_{Rv(F)}.Ext(\Gamma, F \vdash H))nop)\epsilon_1...)\epsilon_2)\epsilon_1,$$
$$(\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(\lambda_{Rv(G)}.Ext(\Gamma, G \vdash H))\epsilon'')\epsilon_1...)\epsilon_2)\epsilon_1)$$
$$\wedge((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.\epsilon'')\epsilon_1...)\epsilon_2)\epsilon_1 \textbf{ realize } G))))$$

Now we can use (2), (3), and the definition of the calculation for *select* to complete the proof.

(b) Similarly for (a).

- Imp-Intro

  Let Γ be $F_1, ..., F_n$ and $x_1, ..., x_m$ be object free variables in $\Gamma, F \vdash G$.

(a) By induction hypothesis

$$\forall x_1, ..., x_m, \epsilon_1, ..., \epsilon_n (R(n, F_i, \epsilon_i) \Rightarrow$$
$$(\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma, F \vdash G))\epsilon_n...)\epsilon_2)\epsilon_1 \textbf{ realize } G) \quad .....(1)$$

By the definition of the program extraction rule

$$Ext(\Gamma \vdash F \Rightarrow G) = (\Pi, \lambda_{Rv(F)}.Ext(\Gamma, F \vdash G))$$

Note that we have

$$\forall x_1, \ldots, x_m, e_1, \ldots, e_n(R(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_n)}.(\Pi, \lambda_{Rv(F)}.Ext(\Gamma, F \vdash G))e_n \ldots)e_2)e_1$$
$$= (\Pi, (\lambda_{Rv(F_2)}.(\ldots(\lambda_{Rv(F_n)}.(\lambda_{Rv(F)}.Ext(\Gamma, F \vdash G)))e_n \ldots)e_2)e_1))$$

Let e_F be an arbitrary Program Term where (e_F **realize** F).

As $Rv(F_i)$ $(0 < i < n+1)$ do not occur in e_F,

$$\forall x_1, \ldots, x_m, e_1, \ldots, e_n(R(n, F_i, e_i) \Rightarrow$$
$$((\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_n)}.(\lambda_{Rv(F)}.Ext(\Gamma, F \vdash G)))e_n \ldots)e_2)e_1)e_F$$
$$= (\lambda_{Rv(F_1)}.(\ldots((\lambda_{Rv(F_n)}.\lambda_{Rv(F)}.Ext(\Gamma, F \vdash G))e_F)e_n \ldots)e_2)e_1)$$

By (1)

$$\forall x_1, \ldots, x_m, e_1, \ldots, e_n(R(n, F_i, e_i) \Rightarrow$$
$$((\lambda_{Rv(F_1)}.(\ldots((\lambda_{Rv(F_n)}.\lambda_{Rv(F)}.Ext(\Gamma, F \vdash G))e_F)e_n \ldots)e_2)e_1 \textbf{ realize } G))$$

(b) Similarly for (a).

- Imp-Elim

  Let Γ be $F_1, \ldots, F_n$ and $x_1, \ldots, x_m$ be object free variables in $\Gamma, F \vdash G$.

(a) By induction hypothesis

$$\forall x_1, \ldots, x_m, e_1, \ldots, e_n(R(n, F_i, e_i) \Rightarrow$$
$$((\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F \Rightarrow G))e_n \ldots)e_2)e_1 \textbf{ realize } F \Rightarrow G)) \quad \ldots(1)$$
$$\forall x_1, \ldots, x_m, e_1, \ldots, e_n(R(n, F_i, e_i) \Rightarrow$$
$$((\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F))e_n \ldots)e_2)e_1 \textbf{ realize } F)) \quad \ldots(2)$$

By the definition of the program extraction rule

$$Ext(\Gamma \vdash G) = (proj(2)(Ext(\Gamma \vdash F \Rightarrow G)))Ext(\Gamma \vdash F)$$

Note that we have

$$\forall x_1, \ldots, x_m, e_1, \ldots, e_n(R(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_n)}.((proj(2)(Ext(\Gamma \vdash F \Rightarrow G)))Ext(\Gamma \vdash F)))e_n \ldots)e_2)e_1$$
$$= ((\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_n)}.((proj(2)(Ext(\Gamma \vdash F \Rightarrow G)))))e_n \ldots)e_2)e_1)$$
$$((\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_n)}.(Ext(\Gamma \vdash F)))e_n \ldots)e_2)e_1))$$

Hence, by (2) and the definition of the realizability interpretation for $\Rightarrow$,

$$\forall x_1, \ldots, x_m, e_1, \ldots, e_n(R(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_n)}.((proj(2)(Ext(\Gamma \vdash F \Rightarrow G)))Ext(\Gamma \vdash F)))e_n \ldots)e_2)e_1$$
$$\textbf{realize } G)$$

(b) Similarly for (a).

- Univ-Intro

  ◦ $\forall$

  Let Γ be $F_1, \ldots, F_n$ and $x, x_1, \ldots, x_m$ be object free variables in $\Gamma \vdash F(x)$.

(a) By induction hypothesis

$$\forall x, x_1, \ldots, x_m, e_1, \ldots, e_n(R(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F(x)))e_n \ldots)e_2)e_1 \textbf{ realize } F(x)) \quad \ldots(1)$$

By the definition of the program extraction rule

$$Ext(\Gamma \vdash \forall x F(x)) = (in, \lambda_x.Ext(\Gamma \vdash F(x)))$$

By (1)

$$\forall x_1, \ldots, x_m, e_1, \ldots, e_n(R(n, F_i, e_i) \Rightarrow$$
$$((\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_n)}.(in, \lambda_x.Ext(\Gamma \vdash F(x))))e_n \ldots)e_2)e_1$$
$$= (in, \lambda_x.e')) \wedge \forall A\ ((\lambda_x.e')A \textbf{ realize } F(A)))$$

(b) Similarly for (a).

◇ $\dot{\forall}$

Similarly for $\forall$.

• Univ-Elim

◇ $\forall$

Let Γ be $F_1, ..., F_n$ and $x_1, ..., x_m$ be object free variables in $\Gamma \vdash \forall x F(x)$.

(a) By induction hypothesis

$$\forall x_1, ..., x_m, e_1, ..., e_n (R(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash \forall x F(x)))e_n...)e_2)e_1 \textbf{ realize } \forall x F(x)) \quad ....(1)$$

By the definition of the program extraction rule

$$Ext(\Gamma \vdash F(a)) = (proj(2)(Ext(\Gamma \vdash \forall x.F(x))))a$$

Note that we have

$$\forall x_1, ..., x_m, e_1, ..., e_n (R(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(proj(2)(Ext(\Gamma \vdash \forall x.F(x))))a)e_n...)e_2)e_1$$
$$= (proj(2)(\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash \forall x.F(x)))e_n...)e_2)e_1)a)$$

Hence, by (1) and the definition of the realizability interpretation for $\forall$,

$$\forall x_1, ..., x_m, e_1, ..., e_n (R(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(proj(2)(Ext(\Gamma \vdash \forall x.F(x))))a)e_n...)e_2)e_1$$
$$\wedge \textbf{ realize } F(a))$$

(b) Similarly for (a).

◇ $\dot{\forall}$

Similarly for $\forall$.

• Exist-Intro

◇ $\exists$

Let Γ be $F_1, ..., F_n, x_1, ..., x_m$ be object free variables in $\Gamma \vdash F(a)$.

(a) By induction hypothesis

$$\forall x, x_1, ..., x_m, e_1, ..., e_n (R(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F(a)))e_n...)e_2)e_1 \textbf{ realize } F(a)) \quad ....(1)$$

By the definition of the program extraction rule

$$Ext(\Gamma \vdash \exists x F(x)) = (out, a, Ext(\Gamma \vdash F(a)))$$

Note that we have

$$\forall x_1, ..., x_m, e_1, ..., e_n (R(n, F_i, e_i) \Rightarrow$$
$$(\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(out, a, Ext(\Gamma \vdash F(a))))e_n...)e_2)e_1$$
$$= (out.(\lambda_{Rv(F_1)}.(...(\dot{\lambda}_{Rv(F_n)}.a)e_n...)e_2)e_1,$$
$$(\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma \vdash F(a)))e_n...)e_2)e_1)).$$

Now we can use (1) and the definition of the realizability interpretation for $\exists$ to complete the proof.

(b) Similarly for (a).

◇ $\dot{\exists}$

Similarly for $\exists$.

• Exist-Elim

◇ $\exists$

Let Γ be $F_1, ..., F_n$ and $y, x_1, ..., x_m$ be object free variables in $\Gamma, F(y) \vdash G$.

(a) By induction hypothesis

$$\forall x_1,...,x_m,e_1,...,e_n(R(n,F_i,e_i)\Rightarrow$$
$$(\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.Ext(\Gamma\vdash\exists xF(x)))e_n...)e_2)e_1 \textbf{ realize } \exists xF(x)) \quad ....(1)$$
$$\forall y,x_1,...,x_m,e_1,...,e_n(R(n,F_i,e_i)\wedge$$
$$((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.e')e_n...)e_2)e_1 \textbf{ realize } F(y))\Rightarrow$$
$$(\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.((\lambda_{Rv(F(y))}.Ext(\Gamma,F(y)\vdash G))e'))e_n...)e_2)e_1$$
$$\textbf{realize } G) \quad ....(2)$$

By the definition of the program extraction rule

$$Ext(\Gamma\vdash G)=(\lambda_{Rv(F(y))}.(\lambda_y.Ext(\Gamma,F(y)\vdash G))$$
$$(proj(2)(Ext(\Gamma\vdash\exists xF(x))))(proj(3)(Ext(\Gamma\vdash\exists xF(x))))$$

By the definition of realizability interpretation of $\exists$, there exists some a such that

$$\forall x_1,...,x_m,e_1,...,e_n(R(n,F_i,e_i)\Rightarrow proj(2)(Ext(\Gamma\vdash\exists xF(x)))=a)$$
$$\forall x_1,...,x_m,e_1,...,e_n(R(n,F_i,e_i)\Rightarrow(proj(3)(Ext(\Gamma\vdash\exists xF(x)))\textbf{ realize } F(a))).$$

Hence, by (1) and (2),

$$\forall x_1,...,x_m,e_1,...,e_n(R(n,F_i,e_i)\Rightarrow$$
$$((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_n)}.(\lambda_{Rv(F(y))}.(\lambda_y.Ext(\Gamma,F(y)\vdash G))$$
$$(proj(2)(Ext(\Gamma\vdash\exists xF(x))))(proj(3)(Ext(\Gamma\vdash\exists xF(x)))))e_n...)e_2)e_1$$
$$\textbf{realize } G))$$

(b) Similarly for (a).

◦ $\dot{\exists}$

Similarly for $\exists$.

• WF-Ind

◦ Case $\dot{\forall}$ occurs in $<\forall>x_1,...,x_n$.

Let Γ be $F_1,...,F_m,w_1,...,w_k$ be object free variables in $\Gamma\vdash\forall x_1,...,x_nF(x_1,...,x_n)$, and assume

$$\nu\equiv\forall y_1,...y_n(<y_1,...,y_n><_{jn}<x_1,...,x_n>\Rightarrow F(y_1,...,y_n))$$
$$\zeta\equiv\Gamma,\forall y_1,...y_n.(<y_1,...,y_n><_{jn}<x_1,...,x_n>\Rightarrow F(y_1,...,y_n))\vdash F(x_1,...,x_n).$$

(a) By induction hypothesis

$$\forall w_1,...,w_k,x_1,...,x_n,e_1,...,e_m(R(m,F_i,e_i)\wedge$$
$$((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_m)}.e')e_m...)e_2)e_1 \textbf{ realize } \nu)$$
$$\Rightarrow((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_m)}.(\lambda_{Rv(\nu)}.Ext(\zeta))e')e_m...)e_2)e_1$$
$$\textbf{realize } F(x_1,...,x_n))) \quad ....(1)$$

By the definition of the program extraction rule

$$Ext(\Gamma\vdash\forall x_1,...,x_nF(x_1,...,x_n))$$
$$=(in,\lambda_{x_1}.(...(in,\lambda_{x_n}.(\mu.(\lambda_{Z,x_1,...,x_n}.(\lambda_{Rv(\nu)}.Ext(\zeta))$$
$$(in,\lambda_{y_1}.(...(in,\lambda_{y_n}.(\Pi,\lambda_x.Zy_1...y_n))...))))x_1...x_n)...))$$

Note that we have

$$(\mu.(\lambda_{Z,x_1,...,x_n}.(\lambda_{Rv(\nu)}.Ext(\zeta))(in,\lambda_{y_1}.(...(in,\lambda_{y_n}.(\Pi,\lambda_x.Zy_1...y_n))...))))x_1...x_n$$
$$=(\lambda_{x_1,...,x_n}.(\lambda_{Rv(\nu)}.Ext(\zeta))(in,\lambda_{y_1}.(...(in,\lambda_{y_n}.(\Pi,\lambda_x.(\mu.(\lambda_{Z,x_1,...,x_n}.(\lambda_{Rv(\nu)}.Ext(\zeta))$$
$$(in,\lambda_{y_1}.(...(in,\lambda_{y_n}.(\Pi,\lambda_x.Zy_1...y_n))...))))y_1...y_n))...)))x_1...x_n \quad ....(2)$$

Let $x_1,...,x_n,w_1,...,w_k$ be arbitrary objects that have values, $e_1,...,e_m$ be arbitrary Program Terms, and assume $R(m,F_i,e_i)$. Then, because of the definition of realizability interpretation for $\forall$ and (2), it suffices to show

$$((\lambda_{Rv(F_1)}.(...(\lambda_{Rv(F_m)}.(\lambda_{x_1,...,x_n}.(\lambda_{Rv(\nu)}.Ext(\zeta))$$
$$(in,\lambda_{y_1}.(...(in,\lambda_{y_n}.(\Pi,\lambda_x.(\mu.(\lambda_{Z,x_1,...,x_n}.(\lambda_{Rv(\nu)}.Ext(\zeta))$$
$$(in,\lambda_{y_1}.(...(in,\lambda_{y_n}.(\Pi,\lambda_x.Zy_1...y_n))...))))y_1...y_n))...)))x_1...x_n)e_m...)e_2)e_1$$
$$\textbf{realize } F(x_1,...,x_n)).$$

As $<_{jn}$ is a well-founded order, we prove this by induction on $<x_1,...,x_n>$.

* **Base Case:**

For all $< y_1, ..., y_n >$, $< x_1, ..., x_n > \leq < y_1, ..., y_n >$ holds. Hence, by realizability interpretations for $\forall$ and $\Rightarrow$, for all Program Term e,

$$((in, \lambda_{y_1}.(...(in, \lambda_{y_n}.(\Pi, e))...)) \textbf{ realize } \nu)$$

Therefore,

$$((\lambda_{Re(F_1)}.(...(\lambda_{Re(F_m)}.(\lambda_{x_1,...,x_n}.(in, \lambda_{y_1}.(...(in, \lambda_{y_n}.(\Pi, \lambda_x.(\mu.(\lambda_{Z,x_1,...,x_n}.(\lambda_{Re(\nu)}.Ext(\zeta))$$
$$(in, \lambda_{y_1}.(...(in, \lambda_{y_n}.(\Pi, \lambda_x.Z y_1 ... y_n))...))))y_1 ... y_n))...)))x_1 ... x_n)e_m...)e_2)e_1 \textbf{ realize } \nu)$$

hence, by (1),

$$((\lambda_{Re(F_1)}.(...(\lambda_{Re(F_m)}.(\lambda_{x_1,...,x_n}.(\lambda_{Re(\nu)}.Ext(\zeta))$$
$$(in, \lambda_{y_1}.(...(in, \lambda_{y_n}.(\Pi, \lambda_x.(\mu.(\lambda_{Z,x_1,...,x_n}.(\lambda_{Re(\nu)}.Ext(\zeta))$$
$$(in, \lambda_{y_1}.(...(in, \lambda_{y_n}.(\Pi, \lambda_x.Z y_1 ... y_n))...))))y_1 ... y_n))...)))x_1 ... x_n)e_m...)e_2)e_1$$
$$\textbf{realize } F(x_1, ..., x_n))$$

* **Induction Case:**

Assume

$$\forall z_1, ..., z_n(< z_1, ..., z_n > \leq < x_1, ..., x_n > \Rightarrow$$
$$((\lambda_{Re(F_1)}.(...(\lambda_{Re(F_m)}.(\lambda_{x_1,...,x_n}.(\lambda_{Re(\nu)}.Ext(\zeta))$$
$$(in, \lambda_{y_1}.(...(in, \lambda_{y_n}.(\Pi, \lambda_x.(\mu.(\lambda_{Z,x_1,...,x_n}.(\lambda_{Re(\nu)}.Ext(\zeta))$$
$$(in, \lambda_{y_1}.(...(in, \lambda_{y_n}.(\Pi, \lambda_x.Z y_1 ... y_n))...))))y_1 ... y_n))...)))z_1 ... z_n)e_m...)e_2)e_1$$
$$\textbf{realize } F(z_1, ..., z_n)))$$

Then, because of (2)

$$\forall z_1, ..., z_n(< z_1, ..., z_n > \leq < x_1, ..., x_n > \Rightarrow$$
$$(\lambda_{Re(F_1)}.(...(\lambda_{Re(F_m)}.(\mu.(\lambda_{Z,x_1,...,x_n}.(\lambda_{Re(\nu)}.Ext(\zeta))$$
$$(in, \lambda_{y_1}.(...(in, \lambda_{y_n}.(\Pi, \lambda_x.Z y_1 ... y_n))...))))z_1 ... z_n)e_m...)e_2)e_1 \textbf{ realize } F(z_1, ..., z_n))$$

Hence,

$$((\lambda_{Re(F_1)}.(...(\lambda_{Re(F_m)}.(in,$$
$$\lambda_{y_1}.(...(in, \lambda_{y_n}.(\Pi, \lambda_x.(\mu.(\lambda_{Z,x_1,...,x_n}.(\lambda_{Re(\nu)}.Ext(\zeta))$$
$$(in, \lambda_{y_1}.(...(in, \lambda_{y_n}.(\Pi, \lambda_x.Z y_1 ... y_n))...))))y_1 ... y_n))...)))e_m...)e_2)e_1 \textbf{ realize } \nu)$$

Hence, by (1),

$$((\lambda_{Re(F_1)}.(...(\lambda_{Re(F_m)}.(\lambda_{x_1,...,x_n}.(\lambda_{Re(\nu)}.Ext(\zeta))$$
$$(in, \lambda_{y_1}.(...(in, \lambda_{y_n}.(\Pi, \lambda_x.(\mu.(\lambda_{Z,x_1,...,x_n}.(\lambda_{Re(\nu)}.Ext(\zeta))$$
$$(in, \lambda_{y_1}.(...(in, \lambda_{y_n}.(\Pi, \lambda_x.Z y_1 ... y_n))...))))y_1 ... y_n))...)))x_1 ... x_n)e_m...)e_2)e_1$$
$$\textbf{realize } F(x_1, ..., x_n))$$

(b) Similarly for (a).

◦ Case $\check{\forall}$ occurs in $< \forall > x_1, ..., x_n$.

Similarly for case $\forall$ occurs in $< \forall > x_1, ..., x_n$.

• Mu-1

Let I be $\mu.\lambda_{\Psi_{(n)},x_1,...,x_n}.G(x_1, ..., x_n, \Psi_{(n)})$, J be $G(b_1, ..., b_n, I)$, ξ be $\Gamma \vdash Ib_1 ... b_n$, Γ be $F_1, ..., F_m$, and $z_1, ..., z_k$ be object free variables in $\Gamma \vdash Ib_1 ... b_n$. Because of the definition of I-formula, no predicate free variables occur in J. Hence, by **Remark** 5.1, it suffices to show (a).

By induction hypothesis

$$\forall z_1, ..., z_k, e_1, ..., e_m(R(m, F_i, e_i)$$
$$\Rightarrow ((\lambda_{Re(F_1)}.(...(\lambda_{Re(F_m)}.Ext(J))e_m...)e_2)e_1 \textbf{ realize } J)) \quad .....(1)$$

By the definition of the program extraction rule

$$Ext(Ib_1 ... b_n) = Ext(J)$$

Let $z_1, ..., z_k$ be arbitrary objects that have values, $e_1, ..., e_m$ be arbitrary Program Terms, and ssume $R(m, F_i, e_i)$. If we define

$S = \{< b_1, ..., b_n >\}$ $\quad A(x_1, ..., x_n, y) \equiv (\lambda_{Rt(F_1)}.(...(\lambda_{Rt(F_m)}.Ext(J))e_m...)e_2)e_1$

$\epsilon' = \lambda_{x_1 ... x_n}.(\lambda_{Rt(F_1)}.(...(\lambda_{Rt(F_m)}.Ext(J))e_m...)e_2)e_1$ $\quad L(x_1, ..., x_n, \Phi) \equiv G(x_1, ..., x_n, I)$

then

$$(\lambda_{Rt(F_1)}.(...(\lambda_{Rt(F_m)}.Ext(J))e_m...)e_2)e_1 = e'a_1...a_n \wedge$$
$$\forall x_1, ..., x_n([I/\Phi]\vartheta(L(x_1, ..., x_n, \Phi)) \equiv G(x_1, ..., x_n, I) \wedge e'x_1...x_n = A(x_1, ..., x_n, e'))$$

and

$$\forall x_1, ..., x_n(< x_1, ..., x_n > \epsilon S \Rightarrow (A(x_1, ..., x_n, Z_\Phi)\ \textbf{realize}\ L(x_1, ..., x_n, \Phi)))$$

$L(x_1, ..., x_n, \Phi)$ has no predicate free variable, hence, by **Remark** 5.1,

$$\forall x_1, ..., x_n(< x_1, ..., x_n > \epsilon S \Rightarrow (A(x_1, ..., x_n, Z_\Phi)\ \textbf{Real}_{<S>}\ L(x_1, ..., x_n, \Phi)))$$

Hence, by **Lemma** 5.1,

$$((\lambda_{Rt(F_1)}.(...(\lambda_{Rt(F_m)}.Ext(J))e_m...)e_2)e_1\ \textbf{realize}\ Ib_1...b_n)$$

TDC is immediate by induction hypothesis.

- Mu-2

Let I be $\mu.\lambda_{\Psi_{(n)}x_1...x_n}.G(x_1, ..., x_n, \Psi_{(n)})$, ξ be $\Gamma \vdash Ib_1...b_n$, Γ be $F_1, ..., F_m$, and $z_1, ..., z_k$ be object free variables in $\Gamma \vdash Ib_1...b_n$. Because of the definition of I-formula, no predicate free variables occur in I. Hence, by **Remark** 5.1, it suffices to show (a).

By induction hypothesis

$$\forall z_1, ..., z_k, e_1, ..., e_m(R(m, F_i, e_i) \Rightarrow$$
$$((\lambda_{Rt(F_1)}.(...(\lambda_{Rt(F_m)}.Ext(\Gamma \vdash Ib_1...b_n))e_m...)e_2)e_1\ \textbf{realize}\ Ib_1...b_n)) \quad ...(1)$$

By the definition of the program extraction rule

$$Ext(\Gamma \vdash G(b_1, ..., b_n, I)) = Ext(\xi)$$

Let $z_1, ..., z_k$ be arbitrary objects, $e_1, ..., e_m$ be arbitrary Program Terms, and assume $R(m, F_i, e_i)$ and

$$e = (\lambda_{Rt(F_1)}.(...(\lambda_{Rt(F_m)}.Ext(\Gamma \vdash Ib_1...b_n))e_m...)e_2)e_1.$$

By **Lemma** 5.1,

◦ There exists some set S such that $< b_1, ..., b_n > \epsilon S$, some wide input context $L(x_1, ..., x_n, \Phi)$, and some Program Term $A(x_1, ..., x_n, y)$ and $e'x_1...x_n$, such that

$$e = e'b_1...b_n \wedge \forall x_1, ..., x_n([I/*][I/\Phi]\vartheta(L(x_1, ..., x_n, \Phi)) \equiv G(x_1, ..., x_n, I) \wedge$$
$$e'x_1...x_n = A(x_1, ..., x_n, e'))$$

and

$$\forall x_1, ..., x_n(< x_1, ..., x_n > \epsilon S \Rightarrow (A(x_1, ..., x_n, Z_\Phi)\ \textbf{Real}_*\ [I/*]L(x_1, ..., x_n, \Phi)))$$

Clearly by **Lemma** 5.1,

$$\forall x_1, ..., x_n(< x_1, ..., x_n > \epsilon S \Rightarrow (e'x_1...x_n\ \textbf{realize}\ Ix_1...x_n))$$

Hence, by **Theorem** 5.2,

$$(A(x_1, ..., x_n, e')\ \textbf{realize}\ [I/*][I/\Phi]\vartheta(L(x_1, ..., x_n, \Phi)))$$

Clearly

$$e = e'b_1...b_n = A(b_1, ..., b_n, e')$$
$$[I/*][I/\Phi]\vartheta(L(x_1, ..., x_n, \Phi)) \equiv G(x_1, ..., x_n, I)$$

so

$$(e\ \textbf{realize}\ G(x_1, ..., x_n, I))$$

TDC is immediate by induction hypothesis.

- Bot-Intro,Neg-Intro,eq 1,eq 2,le

This follows immediately from the definition of the Program Extraction rule and the realizability interpretation.

- Bot-Elim

  Let Γ be $F_1, ..., F_n$ and $x_1, ..., x_m$ be object free variables in $\Gamma \vdash \bot$. Because of the definition of I-formula, no predicate free variables occur in $\bot$. Hence, by **Remark** 5.1, it suffices to show (a). By induction hypothesis

$$\forall x_1, ..., x_m, \epsilon_1, ..., \epsilon_n (R(n, F_i, \epsilon_i) \Rightarrow (\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.Ext(\Gamma \vdash \bot))\epsilon_n...)\epsilon_2)\epsilon_1 \textbf{ realize } \bot)) \quad ....(1)$$

  Hence, by the definition of realizability interpretation,

$$\forall x_1, ..., x_m, \epsilon_1, ..., \epsilon_n (R(n, F_i, \epsilon_i) \Rightarrow (\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.Ext(\Gamma \vdash \bot))\epsilon_n...)\epsilon_2)\epsilon_1 \epsilon \phi))$$

  Hence, it conflicts with $R(n, F_i, \epsilon_i)$.

  Hence, for all $x_1, ..., x_m, \epsilon_1, ..., \epsilon_n$, $R(n, F_i, \epsilon_i)$ does not hold. Therefore, for all Program Term A,

$$\forall x_1, ..., x_m, \epsilon_1, ..., \epsilon_n (R(n, F_i, \epsilon_i) \Rightarrow ((\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.A)\epsilon_n...)\epsilon_2)\epsilon_1 \textbf{ realize } F))$$

  stands. So

$$\forall x_1, ..., x_m, \epsilon_1, ..., \epsilon_n (R(n, F_i, \epsilon_i) \Rightarrow ((\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_n)}.nop)\epsilon_n...)\epsilon_2)\epsilon_1 \textbf{ realize } F))$$

2. • $\exists$

  Let Γ be $F_1, ..., F_k, y_1, ..., y_m$ be object free variables in $\Gamma \vdash \exists x \langle I/\Phi_{(n)} \rangle G(x)$, and

$$\nu \equiv \forall x_1, ..., x_n (H(x_1, ..., x_n) \Rightarrow \Phi_{(n)} x_1 ... x_n),$$
$$\zeta \equiv \Gamma, \forall x_1, ..., x_n (H(x_1, ..., x_n) \Rightarrow \Phi_{(n)} x_1 ... x_n) \vdash \forall x G(x),$$
$$\ell \equiv \Gamma \vdash \exists x (\Phi_{(n)}, \lambda_{x_1 ... x_n}.H(x_1, ..., x_n), I, G(x)), \text{ and}$$
$$I \equiv \mu.\lambda_{\Phi_{(n)} x_1, ..., x_n}.K(x_1, ..., x_n, \Psi a_{11} ... a_{1n}, ..., \Psi a_{m1} ... a_{mn}).$$

  By induction hypothesis

$$\forall y_1, ..., y_m, \epsilon_1, ..., \epsilon_k (R(k, F_i, \epsilon_i) \Rightarrow (\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_k)}.Ext(\Gamma \vdash \exists x \langle I/\Phi_{(n)} \rangle G(x)))\epsilon_k...)\epsilon_2)\epsilon_1 \textbf{ Real}_\kappa \exists x \langle I/\Phi_{(n)} \rangle G(x)) \quad ....(1)$$

$$\forall y_1, ..., y_m, \epsilon_1, ..., \epsilon_k (R(k, F_i, \epsilon_i) \wedge (\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_k)}.\epsilon')\epsilon_k...)\epsilon_2)\epsilon_1 \textbf{ Real}_\kappa \nu) \Rightarrow (\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_k)}.((\lambda_{Rn(\nu)}.Ext(\zeta))\epsilon')\epsilon_k...)\epsilon_2)\epsilon_1 \textbf{ Real}_\kappa \forall x G(x)) \quad ....(2)$$

  By the definition of the program extraction rule

$$Ext(\ell) = (in, \lambda_x.select(if(x = (proj(2)Ext(\exists x \langle I/\Phi_{(n)} \rangle G(x))))), proj(3)Ext(\exists x \langle I/\Phi_{(n)} \rangle G(x)), (proj(2)((\lambda_{Rn(\nu)}.Ext(\zeta))(in, \lambda_{x_1}.(...(in, \lambda_{x_n}.(\Pi, \lambda_y.Z_{\Phi_{(n)}} x_1 ... x_n))...))))x))$$

  Let $y_1, ..., y_m$ be arbitrary objects that have values, $\epsilon_1, ..., \epsilon_k$ be arbitrary Program Terms, and assume $R(k, F_i, \epsilon_i)$. Then, because of the definition of realizability interpretation,

$$((in, \lambda_{x_1}.(...(in, \lambda_{x_n}.\lambda_y.Z_{\Phi_{(n)}} x_1 ... x_n)...)) \textbf{ Real}_\kappa \forall x_1, ..., x_n (H(x_1, ..., x_n) \Rightarrow \Phi_{(n)} x_1 ... x_n))$$

  Hence, by the definition of realizability interpretation for $\exists$ and $\forall$, there exists some ϵ_v such that

$$(\lambda_{Rn(F_1)}.(...(\lambda_{Rn(F_k)}.Ext(\ell))\epsilon_k...)\epsilon_2)\epsilon_1 = (in, \lambda_x.select(if(x = a), \epsilon_u, \epsilon_v)) \wedge (\epsilon_u \textbf{ Real}_\kappa \langle I/\Phi_{(n)} \rangle G(k)) \wedge \forall x.(\epsilon_v \textbf{ Real}_\kappa G(x)) \quad ....(3)$$

  Hence it suffices to show **r**.

  ◦ $\mathbf{r}_1$

  Clear.

  ◦ $\mathbf{r}_2$

  By the definition of $\textbf{Real}_\kappa$,

  Let ϵ'_u, ϵ'_v be the results of replacing $Z_{\Phi_{(n)}}$ with Program Term ϵ on ϵ_u, ϵ_v, where ϵ satisfies $\forall < b_1, ..., b_n > \epsilon S(TIC(eb_1 ... b_n))$, then,

$$tic(\epsilon'_u) \geq min_{<b_1, ..., b_n> \epsilon S}(tic(eb_1 ... b_n)) \quad ....(4)$$

$$tic(c'_i) \geq min_{<b_1,\ldots,b_n>\epsilon S}(tic(eb_1 \ldots b_n)) \qquad \ldots(5)$$

holds. Let e_t be $(\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_k)}.Ext(\ell))e_k\ldots)e_2)e_1$, and e'_t be the result of replacing $Z_{\Phi_{(n)}}$ with ϵ, then because of (3), (4), (5), and the operational semantics of *select*,

$$tic(e'_t) > min_{<b_1,\ldots,b_n>\epsilon S}(tic(eb_1 \ldots b_n))$$

- $\natural$

  Similarly for $\natural$.

3. Let Γ be $F_1, \ldots, F_k$, and $y_1, \ldots, y_n, z_1, \ldots, z_j$ be object free variables in $\Gamma \vdash Ib_1 \ldots b_n$.

$$I \equiv \mu.\lambda_{\Psi_{(n)}x_1 \ldots x_n}.G(x_1, \ldots, x_n, \Psi),$$
$$\varsigma \equiv \Gamma, H(y_1, \ldots, y_n) \vdash \varpi(\Phi_{(n)}.\lambda_{x_1,\ldots,x_n}.H(x_1, \ldots, x_n), I, G(y_1, \ldots, y_n, \Phi_{(n)})),$$
$$\xi \equiv H(y_1, \ldots, y_n), \quad \ell \equiv \Gamma \vdash Ib_1 \ldots b_n \quad , \text{and}$$
$$\nu \equiv \varpi(\Phi_{(n)}.\lambda_{x_1,\ldots,x_n}.H(x_1, \ldots, x_n), I, G(y_1, \ldots, y_n, \Phi_{(n)}))$$

We use **Lemma** 5.1. By induction hypothesis and **Theorem** 5.1.

$$\forall y_1, \ldots, y_n, z_1, \ldots, z_j, e_1, \ldots e_k(R(k, F_i, e_i) \wedge < y_1, \ldots y_n > \epsilon S) \Rightarrow$$
$$((\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_k)}.((\lambda_{Rv(\xi)}.Ext(\varsigma))nop))e_k\ldots)e_2)e_1 \ \mathbf{Real}_\kappa \ \nu)) \qquad \ldots\ldots(1)$$

By the definition of the program extraction rule

$$Ext(\ell) = (\mu.\lambda_{Z_{\Phi_{(n)}}.y_1 \ldots y_n}.(\lambda_{Rv(\xi)}.Ext(\varsigma))nop)a_1 \ldots a_n$$

Let $y_1, \ldots, y_n, z_1, \ldots, z_j$ be arbitrary objects that have values, $e_1, \ldots, e_k$ be arbitrary Program Terms, and assume $R(k, F_i, e_i)$, then,

$$(\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_k)}.Ext(\ell))e_k\ldots)e_2)e_1$$
$$= (\mu.\lambda_{Z_{\Phi_{(n)}}.y_1,\ldots,y_n}.(\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_k)}.((\lambda_{Rv(\xi)}.Ext(\varsigma))nop))e_k\ldots)e_2)e_1)a_1 \ldots a_n$$

Let

$$e''(y_1, \ldots, y_n, Z_{\Phi_{(n)}}) = (\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_k)}.((\lambda_{Rv(\xi)}.Ext(\varsigma))nop))e_k\ldots)e_2)e_1$$

Then

$$< y_1, \ldots, y_n > \epsilon S \Rightarrow (e''(y_2, \ldots, y_n, Z_{\Phi_{(n)}}) \ \mathbf{Real}_\kappa \ \nu)$$

and

$$(\lambda_{Rv(F_1)}.(\ldots(\lambda_{Rv(F_k)}.Ext(\ell))e_k\ldots)e_2)e_1 = (\mu.\lambda_{Z_{\Phi_{(n)}}.y_1,\ldots,y_n}.e''(y_1, \ldots, y_n, Z_{\Phi_{(n)}}))a_1 \ldots a_n$$

holds.

Suppose that

$$e = (\mu.\lambda_{Z_{\Phi_{(n)}}.y_1,\ldots,y_n}.e''(y_1, \ldots, y_n, Z_{\Phi_{(n)}}))a_1 \ldots a_n, \quad L(x_1, \ldots, x_n, \Phi_{(n)}) \equiv \nu,$$
$$e' = (\mu.\lambda_{Z_{\Phi_{(n)}}.y_1,\ldots,y_n}.e''(y_1, \ldots, y_n, Z_{\Phi_{(n)}})), \quad A(x_1, \ldots, x_n, y) = e''(x_1, \ldots, x_n, y),$$

Then

$$e = e'a_1 \ldots a_n \wedge \forall x_1, \ldots, x_n([I/*][I/\Phi]\vartheta(L(x_1, \ldots, x_n, \Phi)) \equiv G(x_1, \ldots, x_n, I)$$
$$\wedge e'x_1 \ldots x_n = A(x_1, \ldots, x_n, e'))$$

Clearly

$$\forall x_1, \ldots, x_n(< x_1, \ldots, x_n > \epsilon S \Rightarrow (A(x_1, \ldots, x_n, Z_\Phi) \ \mathbf{realize} \ [I/*]L(x_1, \ldots, x_n, \Phi)))$$

Now it suffices to show $TDC(\epsilon)$.

By the definition of ϖ, no predicate free variables occur in ν except inside of $\natural$ or $\dot{\natural}$. Hence, by induction hypothesis, let A be an arbitrary Program Term where

$$\forall < b_1, ..., b_n >_{\epsilon S} (TIC(Ab_1...b_n))$$

hold, then

$$\forall y_1, ..., y_n (< y_1, ..., y_n > \epsilon S \Rightarrow (TIC(\epsilon''(y_1, ..., y_n, A)) \wedge min_{<b_1,...,b_n>\epsilon S}(tic(Ab_1...b_n)) < tic(\epsilon''(y_1, ..., y_n, A))))$$

Hence, by

$$tic(\epsilon) = tic((\lambda z_{\Psi_{(n)}} y_1 \ldots y_n . \epsilon'')\epsilon a_1 ... a_n)$$

and induction hypothesis, $1 \leq tic(\epsilon)$ holds.

Repeating the above operation, we can get $tic(\sigma) = \omega$. Hence $TDC(\sigma)$.

□

Appendix 2
Consistency of Logic μ

Proof. Assume Logic μ is inconsistent. Then

$$\vdash \bot$$

is provable. Hence, by **Theorem** 5.3, there exists some Program Term ϵ that can be constructed from the proof, such that

$$(\epsilon \textbf{ realize } \bot)$$

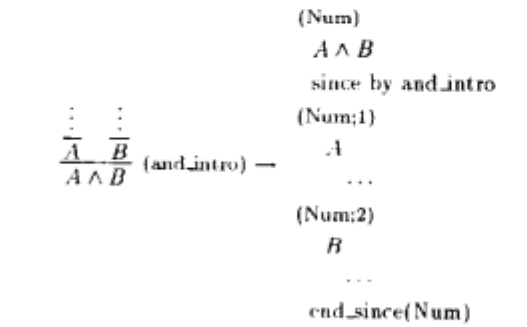
By the definition of realizability interpretation,

$$\epsilon \epsilon \phi$$

This creates a contradiction. Therefore, Logic μ is consistent. □

Appendix 3
Proof

This proof is expressed in PDL (Proof Description Language). We show the relation between PDL and the proof tree, as follows.




```
theorem primes _ filter.
1.
 (μ z:i _ list → prop
  λ p:i _ list ..
  ∀ x:nat
  (¬(x=0) ∧
   (¬div _ list'x'p ∧
    ( ∃ y:int..x=y) /-
    ( ∃ 'k:i _ list..k=cons'x'p ∧ z'k))
   ∨
   div _ list'x'p ∧ z'p)
  ∨
  x=0 ∧ ( ∃ y:string..y='OK'))
 .
 nil
 since by mu _ intro1
1;1.
 ¬(¬(0=0)).
 since by neg _ intro.
1,1;1
 assume ¬(0=0).
 bot.
 since by bot _ intro.
1;1;1;1.
 0=0.
 since by eq _ 1.
 end _ since(1,1,1;1)).
1;1;1;2.
 ¬(0=0).
 since by assumption.
 end _ since((1;1;1;2)).
 end _ since(1,1,1)).
 end _ since(1,1)
1,2.
 let var1:i _ list → prop;
 let var2:i _ list;
 assume ¬(¬(0=0));
 ∃ x:int .
 [ var1.
  λ x:i _ list.. ¬(¬(0=0)),
  μ z:i _ list → prop ..
  λ p:i _ list..
  ∀ x:int ..
  (¬(x=0) ∧
   (¬div _ list'x'p ∧
    ( ∃ y:int..x=y) /-
    ( ∃ 'k:i _ list..k=cons'x'p ∧ z'k))
   ∨
   div _ list'x'p ∧ z'p)
  ∨
  x=0 ∧ ( ∃ y:string..y='OK'),
  λ z:i _ list → prop .
  (¬(x=0) ∧
   (¬div _ list'x'var2 ∧
    ( ∃ y:int..x=y) /-
    ( ∃ 'k:i _ list..k=cons'x'var2 ∧ z'k))
   ∨
   div _ list'x'var2 ∧ z'var2)
   ∨
   x=0 ∧ ( ∃ y:string..y='OK') ].
  since by natural _ intro1.
1;2;1.
  ∃ x:int..
  (¬(x=0) ∧
   (¬div _ list'x'var2 ∧
    ( ∃ y:int..x=y) /-
    ( ∃ 'k:i _ list..
     k=cons'x'var2 ∧
     (μ z:i _ list → prop
      λ p:i _ list..
      ∀ x:int
      (¬(x=0) ∧
       (¬div _ list'x'p ∧
        ( ∃ y:int..x=y) /-
        ( ∃ 'k:i _ list..k=cons'x'p ∧ z'k))
       ∨
       div _ list'x'p ∧ z p)
       ∨
       x=0 ∧ ( ∃ y:string..y='OK'))
      .
      k))
    ∨
    div _ list'x'var2 ∧
    (μ z:i _ list → prop..
     λ p:i _ list..
     ∀ x:int ..
     (¬(x=0) ∧
      (¬div _ list'x'p ∧
       ( ∃ y:int..x=y) /-
       ( ∃ 'k:i _ list..k=cons x p ∧ z'k))
      ∨
      div _ list'x'p ∧ z'p)
      ∨
      x=0 ∧ ( ∃ y:string..y='OK'))
     .
     var2)
    ∨
   x=0 ∧ ( ∃ y:string..y='OK').
  since by exists _ intro.
1,2;1;1.
   (¬(0=0) ∧
    (¬div _ list'0'var2 ∧
     ( ∃ y:int..0=y) /-
     ( ∃ 'k:i _ list..
      k=cons'0'var2 ∧
      (μ z:i _ list → prop
       λ p:i _ list..
       ∀ x:int ..
       (¬(x=0) ∧
        (¬div _ list'x'p ∧
         ( ∃ y:int..x=y) /-
         ( ∃ 'k:i _ list..k=cons'x'p ∧ z'k))
        ∨
        div _ list'x'p ∧ z'p)
        ∨
        x=0 ∧ ( ∃ y:string..y='OK'))
       .
       k))
     ∨
     div _ list'0'var2 ∧
     (μ z:i _ list → prop..
      λ p:i _ list..
      ∀ x:int ..
      (¬(x=0) ∧
       (¬div _ list'x'p ∧
        ( ∃ y:int..x=y) /-
        ( ∃ 'k:i _ list..k=cons'x'p ∧ z'k))
       ∨
       div _ list'x'p ∧ z'p)
       ∨
       x=0 ∧ ( ∃ y:string..y='OK'))
      .
      var2)
```

```
    ∨
    0=0 ∧ ( ∃ y:string. y='OK').
    since by or _ elim.
1;2;1;1;2:
    0=0 ∨ ¬(0=0).
    since by univ2 _ elim
1;2;1;1;1;1:
    ∀ 'x:int..x=0 ∨ ¬(x=0).
    since by 'cap#equal _ lemma'.
    end _ since({1;2;1;1;1;1}).
   end _ since({1;2;1;1;1}).
1;2;1;1;2
   assume 0=0;
   (¬(0=0) ∧
    (¬div _ list'0'var2 ∧
     ( ∃ y:int. 0=y) /-
     ( ∃ 'k:i _ list
      k=cons'0'var2 ∧
      (μ z:i _ list → prop.
       λ p:i _ list
       ∀ x:int.
       (¬(x=0) ∧
        (¬div _ list'x'p ∧
         ( ∃ y:int. x=y) /-
         ( ∃ 'k:i _ list.k=cons'x'p ∧ z'k))
        ∨
        div _ list'x'p ∧ z'p)
       ∨
       x=0 ∧ ( ∃ y:string.y='OK'))
      z
      k))
    ∨
    div _ list'0'var2 ∧
    (μ z:i _ list → prop.
     λ p:i _ list.
     ∀ x:int
     (¬(x=0) ∧
      (¬div _ list'x'p ∧
       ( ∃ y:int. x=y) /-
       ( ∃ 'k:i _ list.k=cons'x'p ∧ z'k))
      ∨
      div _ list'x'p ∧ z'p)
     ∨
     x=0 ∧ ( ∃ y:string.y='OK'))
     '
    var2)
    ∨
   0=0 ∧ ( ∃ y:string.y='OK')
   since by or _ intro _ r.
1;2;1;1;2;1:
    0=0 ∧ ( ∃ y:string.y='OK').
    since by and _ intro.
1;2;1;1;2;1;1:
    0=0.
    since by assumption.
    end _ since({1;2;1;1;2;1;1}).
1;2;1;1;2;1;2:
    ∃ y:string.y='OK'.
    since by exists _ intro.
1;2;1;1;2;1;2;1:
    'OK'='OK'.
    since by eq _ 1.
    end _ since({1;2;1;1;2;1;2;1}).
   end _ since({1;2;1;1;2;1;2}).
  end _ since({1;2;1;1;2;1}).
 end _ since({1;2;1;1;2}).
1;2;1;1;3:
 assume ¬(0=0);
 (¬(0=0) ∧
  (¬div _ list'0'var2 ∧
   ( ∃ y:int.0=y) /-
   ( ∃ 'k:i _ list.
    k=cons'0'var2 ∧
    (μ z:i _ list → prop.
     λ p:i _ list.
     ∀ x:int.
     (¬(x=0) ∧
      (¬div _ list'x'p ∧
       ( ∃ y:int.x=y) /-
       ( ∃ 'k:i _ list.k=cons'x'p ∧ z'k))
      ∨
      div _ list'x'p ∧ z'p)
     ∨
     x=0 ∧ ( ∃ y:string.y='OK'))
    var2)
   ∨
  0=0 ∧ ( ∃ y:string.y='OK')
  since by bot _ elim.
1;2;1;1;3;1
   bot.
   since by bot _ intro.
1;2;1;1;3;1;1:
   0=0.
   since by eq _ 1.
   end _ since({1;2;1;1;3;1;1}).
1;2;1;1;3;1;2
   ¬(0=0).
   since by assumption
   end _ since({1;2;1;1;3;1;2}).
  end _ since({1;2;1;1;3;1}).
 end _ since({1;2;1;1;3}).
end _ since({1;2;1;1}).
1;2;2
 assume ∀ 'x:i _ list.. (¬(¬(0=0)) ⇒ var1'x).
 ∀ x:int.
 (¬(x=0) ∧
  (¬div _ list'x'var2 ∧
   ( ∃ y:int..x=y) /-
   ( ∃ 'k:i _ list..k=cons'x'var2 ∧ var1'k))
  ∨
  div _ list'x'var2 ∧ var1'var2)
  ∨
 x=0 ∧ ( ∃ y:string.y='OK').
 since by univ _ intro.
1;2;2;1:
  let v _ 1:int.
  (¬(var3=0) ∧
   (¬div _ list'var3'var2 ∧
    ( ∃ y:int..var3=y) /-
    ( ∃ 'k:i _ list.k=cons'var3'var2 ∧ var1'k))
    ∨
   div _ list'var3'var2 ∧ var1'var2)
   ∨
  var3=0 ∧ ( ∃ y:string.y='OK').
  since by or _ elim.
1;2;2;1;1:
   var3=0 ∨ ¬(var3=0).
   since by univ2 _ elim.
1;2;2;1;1;1:
   ∀ 'x:int..x=0 ∨ ¬(x=0).
   since by 'cap#equal _ lemma'.
   end _ since({1;2;2;1;1;1}).
  end _ since({1;2;2;1;1}).
1;2;2;1;2:
   assume var3=0;
   (¬(var3=0) ∧
   (¬div _ list'var3'var2 ∧
    ( ∃ y:int..var3=y) /-
    ( ∃ 'k:i _ list..k=cons'var3'var2 ∧ var1'k))
    ∨
   div _ list'var3'var2 ∧ var1'var2)
   ∨
  var3=0 ∧ ( ∃ y:string.y='OK').
  since by or _ intro _ r.
```

```
1:2,2,1:2;1.
    var3=0 ∧ ( ∃ y:string..y='OK').
    since by and _ intro.
1:2,2,1:2:1;1.
    var3=0.
    since by assumption.
    end _ since({1;2;2;1;2;1;1}).
1:2,2,1:2;1;2
    ∃ y:string..y='OK'.
    since by exists _ intro.
1:2,2,1:2;1;2;1.
    'OK'='OK'.
    since by eq _ 1.
    end _ since({1,2,2,1,2;1,2,1}).
   end _ since({1,2,2,1:2;1;2}).
  end _ since({1;2;2;1;2;1}).
 end _ since({1;2,2,1:2}).
1:2,2,1:3.
  assume ¬(var3=0);
  (¬(var3=0) ∧
   (¬div _ list'var3'var2 ∧
    ( ∃ y:int..var3=y) /\
    ( ∃ 'k i _ list..k=cons'var3'var2 ∧ var1'k))
   ∨
   div _ list'var3'var2 ∧ var1'var2)
  ∨
  var3=0 ∧ ( ∃ y:string..y='OK').
  since by or _ elim.
1:2,2,1:3;1.
   div _ list'var3'var2 ∨ ¬div _ list'var3'var2
   since by univ2 _ elim.
1:2,2,1:3;1;1.
    ∀ 'y:int..div _ list'y'var2 ∨ ¬div _ list'y'var2.
   since by univ2 _ elim
1:2,2,1:3;1;1;1
    ∀ 'x:i _ list..
    ∀ 'y:int..div _ list'y'x ∨ ¬div _ list'y'x
    since by 'cap@div _ list _ lemma'.
    end _ since({1;2;2;1;3;1;1;1}).
   end _ since({1,2,2,1;3,1,1}).
  end _ since({1,2,2,1;3;1}).
1:2,2,1:3;2
   assume div _ list'var3'var2;
   (¬(var3=0) ∧
    (¬div _ list'var3'var2 ∧
     ( ∃ y:int..var3=y) /\
     ( ∃ 'k:i _ list..k=cons'var3'var2 ∧ var1'k))
    ∨
    div _ list'var3'var2 ∧ var1'var2)
   ∨
   var3=0 ∧ ( ∃ y:string..y='OK').
   since by or _ elim.
1:2,2,1:3;2,1.
    var3=0 ∨ ¬(var3=0).
    since by univ2 _ elim.
1:2,2,1:3;2,1;1.
    ∀ 'x:int..x=0 ∨ ¬(x=0).
    since by 'cap@equal _ lemma'
    end _ since({1;2;2;1;3;2;1;1}).
   end _ since({1;2;2;1;3;2;1}).
1:2,2,1:3;2;2.
   assume var3=0;
   (¬(var3=0) ∧
    (¬div _ list'var3'var2 ∧
     ( ∃ y:int..var3=y) /\
     ( ∃ 'k i _ list..k=cons'var3'var2 ∧ var1'k))
    ∨
    div _ list'var3'var2 ∧ var1'var2)
   ∨
   var3=0 ∧ ( ∃ y:string..y='OK').
   since by or _ intro _ r.
1:2;2,1;3;2;2;1.
    var3=0 ∧ ( ∃ y:string..y='OK')
    since by and _ intro.
1:2,2,1;3;2;2;1;1
    var3=0
    since by assumption.
    end _ since({1;2;2;1;3;2;2;1;1}).
1:2,2,1;3;2;2;1;2.
    ∃ y:string..y='OK'.
    since by exists _ intro.
1:2;2,1;3;2;2;1;2;1.
    'OK'='OK'.
    since by eq _ 1.
    end _ since({1,2;2;1;3;2;2;1,2;1}).
   end _ since({1,2;2;1;3;2;2;1:2}).
  end _ since({1,2;2;1;3;2;2;1}).
 end _ since({1,2,2,1,3;2,2}).
1:2;2;1;3;2;3.
   assume ¬(var3=0);
   (¬(var3=0) ∧
    (¬div _ list'var3'var2 ∧
     ( ∃ y:int..var3=y) /\
     ( ∃ 'k i _ list..k=cons'var3'var2 ∧ var1'k))
    ∨
    div _ list'var3'var2 ∧ var1'var2)
   ∨
   var3=0 ∧ ( ∃ y:string..y='OK').
   since by or _ intro _ l.
1:2,2,1;3;2;3;1.
    ¬(var3=0) ∧
    (¬div _ list'var3'var2 ∧
     ( ∃ y:int..var3=y) /\
     ( ∃ 'k i _ list..k=cons'var3'var2 ∧ var1'k))
    ∨
    div _ list'var3'var2 ∧ var1'var2.
   since by and _ intro.
1,2,2,1;3;2;3;1;1.
    ¬(var3=0).
    since by assumption.
    end _ since({1,2,2,1,3,2,3,1;1}).
1,2,2,1;3;2;3;1;2
    (¬div _ list'var3'var2 ∧
     ( ∃ y:int..var3=y) /\
     ( ∃ 'k i _ list..k=cons'var3'var2 ∧ var1'k))
    ∨
    div _ list'var3'var2 ∧ var1'var2.
   since by or _ intro _ r
1,2,2,1;3;2;3;1;2;1.
    div _ list'var3'var2 ∧ var1'var2.
    since by and _ intro.
1,2,2,1;3;2;3;1;2;1;1
    div _ list'var3'var2.
    since by assumption.
    end _ since({1,2;2;1;3;2;3;1,2;1;1}).
1;2;2;1;3;2;3;1;2;1;2
    var1'var2
    since by imp _ elim.
1,2,2,1;3;2;3;1;2;1;2;1.
    ¬(¬(0=0)) ⇒ var1'var2.
    since by univ2 _ elim
1,2,2,1;3;2;3;1;2;1;2;1;1
    ∀ 'x:i _ list.. (¬(¬(0=0)) ⇒ var1'x)
    since by assumption.
    end _ since({1,2,2,1;3;2;3;1;2;1;2;1;1}).
   end _ since({1,2;2;1;3;2;3;1;2;1;2;1}).
1;2;2;1;3;2;3;1;2;1;2;2
    ¬(¬(0=0)).
    since by assumption.
    end _ since({1;2;2;1;3;2;3;1;2;1;2}).
   end _ since({1;2;2;1;3;2;3;1;2;1}).
  end _ since({1;2;2;1;3;2;3;1;2}).
 end _ since({1,2;2;1;3;2;3,1;2}).
end _ since({1,2;2;1;3;2;3,1}).
end _ since({1,2,2,1;3;2;3}).
end _ since({1,2,2,1;3;2}).
1:2;2;1;3;3.
   assume ¬div _ list'var3'var2;
   (¬(var3=0) ∧
    (¬div _ list'var3'var2 ∧
     ( ∃ y:int..var3=y) /\
     ( ∃ 'k i _ list..k=cons'var3'var2 ∧ var1'k))
    ∨
    div _ list'var3'var2 ∧ var1'var2)
   ∨
   var3=0 ∧ ( ∃ y:string..y='OK').
   since by or _ intro _ l.
1,2,2,1;3;3;1.
    ¬(var3=0) ∧
    (¬div _ list'var3'var2 ∧
     ( ∃ y:int..var3=y) /\
     ( ∃ 'k i _ list..k=cons'var3'var2 ∧ var1'k))
    ∨
    div _ list'var3'var2 ∧ var1'var2.
```

```
      since by and _ intro.
1;2;2;1;3;3;1;1.
      ¬(var3=0).
      since by assumption.
      end _ since(1;2;2;1;3;3;1;1)).
1;2;2;1;3;3;1;2.
      (¬div _ list`var3`var2 ∧
       ( ∃ y:int..var3=y) /-
       ( ∃ 'k:i _ list..k=cons`var3`var2 ∧ var1`k))
      ∨
      div _ list`var3`var2 ∧ var1`var2.
      since by or _ intro _ 1.
1;2;2;1;3;3;1;2;1.
       ¬div _ list`var3`var2 ∧
       ( ∃ y:int..var3=y) /-
       ( ∃ 'k:i _ list..k=cons`var3`var2 ∧ var1`k)
       since by and _ intro.
1;2;2;1;3;3;1;2;1;1.
        ¬div _ list`var3`var2.
        since by assumption.
        end _ since(1;2;2;1;3;3;1;2;1;1)).
1;2;2;1;3;3;1;2;1;2.
        ( ∃ y:int..var3=y) /-
        ( ∃ 'k:i _ list..k=cons`var3`var2 ∧ var1`k).
        since by and _ s _ intro.
1;2;2;1;3;3;1;2;1;2;1.
         ∃ y:int..var3=y
         since by exists _ intro.
1;2;2;1;3;3;1;2;1;2;1;1.
          var3=var3.
          since by eq _ 1.
          end _ since(1;2;2;1;3;3;1;2;1;2;1;1)).
         end _ since(1;2;2;1;3;3;1;2;1;2;1)).
1;2;2;1;3;3;1;2;1;2;2.
         ∃ 'k:i _ list..k=cons`var3`var2 ∧ var1`k.
         since by exists2 _ intro.
1;2;2;1;3;3;1;2;1;2;2;1.
          cons`var3`var2=cons`var3`var2 ∧
          var1`(cons`var3`var2)
          since by and _ intro.
1;2;2;1;3;3;1;2;1;2;2;1;1.
           cons`var3`var2=cons`var3`var2.
           since by eq _ 1.
           end _ since(1;2;2;1;3;3;1;2;1;2;2;1;1))
1;2;2;1;3;3;1;2;1;2;2;1;2.
           var1`(cons`var3`var2).
           since by imp _ elim.
1;2;2;1;3;3;1;2;1;2;2;1;2;1.
           ¬(¬(0=0)) ⇒ var1`(cons`var3`var2).
           since by univ2 _ elim.
1;2;2;1;3;3;1;2;1;2;2;1;2;1;1.
            ∀ 'x:i _ list..(¬(¬(0=0)) ⇒ var1`x)
            since by assumption.
            end _ since(1;2;2;1;3;3;1;2;1;2;2;1;2;1;1)).
           end _ since(1;2;2;1;3;3;1;2;1;2;2;1;2;1)
1;2;2;1;3;3;1;2;1;2;2;1;2;2.
           ¬(¬(0=0)).
           since by assumption.
           end _ since(1;2;2;1;3;3;1;2;1;2;2;1;2;2)).
          end _ since(1;2;2;1;3;3;1;2;1;2;2;1;2)).
         end _ since(1;2;2;1;3;3;1;2;1;2;2;1)).
        end _ since(1;2;2;1;3;3;1;2;1;2;2)).
       end _ since(1;2;2;1;3;3;1;2;1;2)).
      end _ since(1;2;2;1;3;3;1;2;1)).
     end _ since(1;2;2;1;3;3;1;2)).
    end _ since(1;2;2;1;3;3;1)).
   end _ since(1;2;2;1;3;3)).
  end _ since(1;2;2;1;3)).
 end _ since(1;2;2;1)).
end _ since(1;2;2)).
end _ since(1;2)).
end _ since(1).
```

Appendix 4
Program

The syntax of the follow Program Term has been changed as follows, because of our implementation circumstances.

- $AB \rightarrow app(A, B)$
- $\lambda_x.A \rightarrow lambda(x, A)$

```
app(
 mu
  lambda(
   pr _ mu,
   lambda(
    x33,
    app(
     lambda(
      subprf3,
      ( in,
        lambda(
         w,
         select(
          if(w=app(proj(2),
           (out,0,(x,(nop,(para,nop,(out,'OK',nop))))))),
          app(proj(3),(out,0,(r,(nop,(para,nop,(out,'OK',nop))))))),
         app(
          app(
           proj(2),
           app(
            lambda(
             subprf2,
             ( in,
               lambda(
                x36,
                select(
                 app(
                  proj(1),
                  app(app(proj(2),
                   (in,lambda(q,(if(q=0),(nop,nop))))),x36)
                 ),
                 app(
                  lambda(subprf3,
                   (x,(nop,(para,subprf3,(out,'OK',nop))))),
                  app(
                   proj(1),
                   app(
                    proj(2),
                    app(app(proj(2),
                     (in,lambda(q,(if(q=0),(nop,nop))))),x36)
                    )
                   )
                  ),
                 app(
                  lambda(
                   subprf4,
                   ( l,
                     ( ( para,
                         subprf4,
                         select(
                          app(
                           proj(1),
                           app(
                            app(
                             proj(2),
                             app(
                              app(
                               proj(2),
                               ( in,
                                 lambda(
                                  q,
                                  ( in,
                                    lambda(
                                     qt,
                                     ( app(
                                        app(
                                         mu
                                         lambda(
                                          c,
                                          lambda(
                                           d,
                                           lambda(
                                            e,
                                            select(
                                             if(e=( )),
                                             r,
                                             select(
                                              if(d/car(e)*car(e)=d),
                                              ),
                                              app(app(c,d),cdr(e))
                                             )
                                            )
                                           )
                                          )
                                         ),
                                        qt
```

```
                ),
                q
               ),
               (nop,nop) )
              ) )
             ) )
           ),
           x33
          )
         ),
         x36
        )
       ),
       app(
        lambda(
         subprf5,
         ( r,
          ( nop,
           ( para,
            subprf5,
            app(
app(proj(2),app(app(proj(2),subprf2),x33)),
             subprf1
            ) ) ) )
         ),
         app(
          proj(1),
          app(
           proj(2),
           app(
            app(
             proj(2),
             app(
              app(
               proj(2),
               ( in,
                lambda(
                 q,
                 ( in,
                  lambda(
                   qt,
                   ( app(
                     app(
                      mu
                      lambda(
                       c,
                       lambda(
                        d,
                        lambda(
                         e,
                         select(
                          if(e=( )),
                          r,
                          select(
                           if(d/car(e)*car(e)=d),
                           ),
                           app(app(c,d),cdr(e))
                          )
                         )
                        )
                       ),
                      qt
                     ),
                     q
                    ),
                    (nop,nop) )
                  ) )
                ) )
              ),
              x33
             )
            ),
            x36
           )
          )
         )
        ),
        app(
         lambda(
          subprf6,
          ( ),
           ( ( para,
              subprf6,
              ( seq,
               (out,x36,nop),
               ( nonout,
                cons(x36,x33),
                ( para,
                 nop,
                 app(
                  app(
                   proj(2),
                   app(app(proj(2),subprf2),cons(x36,x33))
                  ),
                  subprf1
                 ) ) ) ) ),
            nop ) )
         ),
         app(
          proj(2),
          app(
           proj(2),
           app(
            app(
             proj(2),
             app(
              app(
               proj(2),
               ( in,
                lambda(
                 q,
                 ( in,
                  lambda(
                   qt,
                   ( app(
                     app(
                      mu
                      lambda(
                       c,
                       lambda(
                        d,
                        lambda(
                         e,
                         select(
                          if(e=( )),
                          r,
                          select(
                           if(d/car(e)*car(e)=d),
                           ),
                           app(app(c,d),cdr(e))
                          )
                         )
                        )
                       ),
                      qt
                     ),
                     q
                    ),
                    (nop,nop) )
                  ) )
                ) )
              ),
              x33
             )
            ),
            x36
           )
          )
         )
        ),
        nop ) )
      ),
      app(
       proj(2),
       app(
        proj(2),
        app(app(proj(2),
         (in,lambda(q,(if(q=0),(nop,nop+1)))),x36)
       )
      )
     )
    )
   ),
   (in,lambda(val,(pi,lambda(val2,app(pr_  mu,val))))))
  )
 ),
 w
)
)
) )
),
nop
)
)
),
nil
).
```