PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT. © ICOT, 1992

Realizability Interpretation of Coinductive Definitions and Program Synthesis with Streams

Makoto Tatsuta

Research Institute of Electrical Communication,
Tohoku University,
2-1-1 Katahira, Sendai 980, JAPAN
e-mail: tatsuta@riec.tohoku.ac.jp

Abstract

The main aim of this paper is to construct a logic by which properties of programs can be formalized for verification, synthesis and transformation of programs.

This paper has 2 main points. One point is realizability interpretation of coinductive definitions of predicates. The other point is an extraction of programs which treat streams.

An untyped predicative theory $\mathbf{TID}_\nu$ is presented, which has the facility of coinductive definitions of predicates and is based on a constructive logic. Properties defined by the greatest fixed point, such as streams and the extensional equality of streams, can be formalized by the facility of coinductive definitions of predicates in $\mathbf{TID}_\nu$.

q-realizability interpretation for $\mathbf{TID}_\nu$ is defined and the realizability interpretation is proved to be sound.

By the realizability interpretation, a program which treats streams can be extracted from a proof of its specification in $\mathbf{TID}_\nu$. General program extraction theorem and stream program extraction theorem are presented.

1 Introduction

Our main aim is to construct a logic by which we can formalize properties of programs for verification, synthesis and transformation of programs. In this paper, we concentrate on formalization of programs with streams and present a theory $\mathbf{TID}_\nu$.

Coinductive definitions are very important for this purpose. Properties of streams are represented semantically by the greatest fixed point. The predicate representing what a stream is and the extensional equality of streams are defined semantically by the greatest fixed point. These properties defined by the greatest fixed point can be formalized by coinductively defined predicates and coinduction.

μ-calculus has been studied to formalize programs with streams for verification [3]. μ-calculus has the facility of coinductive definitions of predicates and coinduction and is based on classical logic.

In this paper, we present a theory $\mathbf{TID}_\nu$, which has the facility of coinductive definitions of predicates and coinduction and is based on a constructive logic. By these facilities we can formalize properties of programs with streams in $\mathbf{TID}_\nu$.

Our theory $\mathbf{TID}_\nu$ is based on a constructive logic because we want to use the facility of program extraction by realizability for $\mathbf{TID}_\nu$. Program extraction is one of the benefits we get when we use a constructive formal theory to formalize properties of programs. Program extraction is to get a program from a constructive proof of its specification formula. One method of program extraction is to use realizability interpretation. In $\mathbf{PX}$[4], for example, a LISP program is extracted from a proof of its specification formula by realizability interpretation.

By the facility of coinductive definitions of predicates and realizability interpretation, we can synthesize programs with streams naturally in $\mathbf{TID}_\nu$ using theorem proving techniques.

This paper has 2 main points. One point is realizability interpretation of coinductive definitions. The other point is an extraction of programs with streams.

We present an untyped predicative theory $\mathbf{TID}_\nu$, which has coinductive definitions of predicates and is based on a constructive logic. We define q-realizability interpretation of $\mathbf{TID}_\nu$. We show that the realizability interpretation is sound. We present general program extraction theorem and stream program extraction theorem.

The soundness proof is based on the early version of this paper [8]. The soundness theorem was proved also in [5]. Both works are independent.

In Section 2, we define a theory $\mathbf{TID}_\nu$. In Section 3, we briefly explain how useful the facility of coinductive definitions of predicates is to formalize streams. In Section 4, we discuss a model of $\mathbf{TID}_\nu$ and prove its consistency. In Section 5, we present q-realizability interpretation of $\mathbf{TID}_\nu$ and prove the soundness theorem. In Section 6, we give general program extraction theorem, stream program extraction theorem for $\mathbf{TID}_\nu$ and an example of program synthesis.

2 Theory $\mathbf{TID}_\nu$

We present a theory $\mathbf{TID}_\nu$ in this section. It is the same as Beeson's **EON** [1] except for the axioms of coinductive definitions of predicates.

In this paper, we choose combinators as the target programming language for simplicity since we want to concentrate on the topic of coinductive definitions of predicates. We suppose that the evaluation strategy of combinators is lazy or call by name because we represent a stream by an infinite list, which is a non-terminating term. We omit also the formalization of the lazy or call by name evaluation strategy in $\mathbf{TID}_\nu$ for simplicity.

Definition 2.1. (Language of $\mathbf{TID}_\nu$)

The language of $\mathbf{TID}_\nu$ is based on a first order language but extended for coinductive definitions of predicates.

The constants are:

$$\mathbf{K},\quad \mathbf{S},\quad \mathbf{p},\quad \mathbf{p}_0,\quad \mathbf{p}_1,\quad 0,\quad \mathbf{s}_\mathrm{N},\quad \mathbf{p}_\mathrm{N},\quad \mathbf{d}.$$

We choose combinators as a target programming language for simplicity. **K** and **S** mean the usual basic combinators. We have natural numbers as primitiyes, which are given by 0, a successor function $\mathbf{s}_\mathrm{N}$ and a predecessor function $\mathbf{p}_\mathrm{N}$. We also have paring functions $\mathbf{p}$, $\mathbf{p}_0$ and $\mathbf{p}_1$ as built-in, which correspond to cons, car and cdr in LISP respectively. **d** is a combinator judging equality of natural numbers and corresponds to an if-then-else statement in a usual programming language.

We have only one function symbol:

App

whose arity is 2. It means a functional application of combinators.

Terms are defined in the same way as for a usual first order logic. For terms s, t, we abbreviate $\mathbf{App}(s,t)$ as st. For terms s, t, we also use an abbreviation $\langle s,t\rangle \equiv \mathbf{p}st$, $t_0 \equiv \mathbf{p}_0 t$ and $t_1 \equiv \mathbf{p}_1 t$.

The predicate symbols are:

$$\bot,\quad \mathrm{N},\quad =.$$

We have predicate variables, which a first order language does not have. The predicate variables are:

$$X, Y, Z, \ldots, X^*, Y^*, Z^*, \ldots.$$

Each predicate variable has a fixed arity.

We use an abbreviation $\lambda x.t$ which is constructed by combinators in the usual way. We also abbreviate $\mathrm{Y}(\lambda x.t)$ as $\mu x.t$ where $\mathrm{Y} \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$.

Definition 2.2. (Formula)

We define a formula A, a set $S_+(A)$ of predicate variables which occur positively in A and a set $S_-(A)$ of predicate variables which occur negatively in A.

1. If a, b are terms,
   $\bot,\quad \mathrm{N}(a),\quad a = b$
   are formulas. Then
   $S_+(\bot) = S_-(\bot) = \phi,$
   $S_+(\mathrm{N}(a)) = S_-(\mathrm{N}(a)) = \phi,$
   $S_+(a=b) = S_-(a=b) = \phi.$

2. If X is a predicate variable whose arity is n, $X(x_1, \ldots, x_n)$ is a formula and
   $S_+(X(x_1,\ldots,x_n)) = \{X\},$
   $S_-(X(x_1,\ldots,x_n)) = \phi.$

3. $A \,\&\, B$, $A \vee B$, $A \to B$, $\forall x A$, $\exists x A$ are formulas if A and B are formulas in the same way as a first order language. Then
   $S_+(A \,\&\, B) = S_+(A \vee B) = S_+(A) \cup S_+(B),$
   $S_-(A \,\&\, B) = S_-(A \vee B) = S_-(A) \cup S_-(B),$
   $S_+(A \to B) = S_-(A) \cup S_+(B),$
   $S_-(A \to B) = S_+(A) \cup S_-(B),$
   $S_+(\forall x A) = S_+(\exists x A) = S_+(A),$
   $S_-(\forall x A) = S_-(\exists x A) = S_-(A).$

4. $(\nu X.\lambda x_1 \ldots x_n.A)(t_1, \ldots, t_n)$ is a formula where X is a predicate variable whose arity is n, A is a formula, $t_1, \ldots, t_n$ are terms and X is not in $S_-(A)$. Then
   $S_+((\nu X.\lambda x_1 \ldots x_n.A)(t_1,\ldots,t_n)) =$
   $\quad S_+(A) - \{X\},$
   $S_-((\nu X.\lambda x_1 \ldots x_n.A)(t_1,\ldots,t_n)) = S_-(A).$

$\bot$ means contradiction. $\mathrm{N}(a)$ means that a is a natural number. $a = b$ means that a equals to b.

The last case corresponds to coinductively defined predicates. Remark that X and $x_1, \ldots, x_n$ may occur freely in A. The intuitive meaning of a formula

$$(\nu X.\lambda x_1 \ldots x_n.A(X, x_1, \ldots, x_n))(t_1, \ldots, t_n)$$

is as follows: Let P be a predicate of arity n such that P is the greatest solution of an equation

$$P(x_1, \ldots, x_n) \leftrightarrow A(P, x_1, \ldots, x_n).$$

Then $(\nu X.\lambda x_1 \ldots x_n.A(X, x_1, \ldots, x_n))(t_1, \ldots, t_n)$ means $P(t_1, \ldots, t_n)$ intuitively.

We abbreviate a sequence as a bold type symbol, for example, $x_1, \ldots, x_n$ as $\mathbf{x}$.

Example 2.3.

We give an example of a formula. We assume the arity of a predicate variable P is 1. Then

$$(\nu P.\lambda x.x = \langle x_0, x_1\rangle \,\&\, x_0 = 0 \,\&\, P(x_1))(x)$$

is a formula.

Among many axioms and inference rules of $\mathbf{TID}_\nu$, we discuss only inference rules of coinductive definitions of predicates here. The rest of axioms and inference rules are almost the same as **EON** [1] and we only list them in Appendix A.

Definition 2.4. (Coinductive Definitions)

Let $\nu \equiv \nu P.\lambda \mathbf{x}.A(P)$ where $\mathbf{x}$ is a sequence of variables whose length is the same as the arity of a predicate variable P and $A(P)$ is a formula displaying all the occurrences of P in a formula A. Suppose that $C(\mathbf{x})$ is a formula displaying all the occurrences of variables $\mathbf{x}$ in the formula.

We have the following axioms:

$$\forall \mathbf{x}(\nu(\mathbf{x}) \to A(\nu)), \qquad (\nu 1)$$

$$\forall \mathbf{x}(C(\mathbf{x}) \to A(C)) \to \forall \mathbf{x}(C(\mathbf{x}) \to \nu(\mathbf{x})). \qquad (\nu 2)$$

$\nu P.\lambda x.A(P)$ means the greatest fixed point of the function from a predicate P to a predicate $\lambda x.A(P)$.

We define a theory $\mathbf{TID}^-$ as a theory $\mathbf{TID}_\nu$ except for the 2 axioms of coinductive definitions of predicates.

3 Coinductive Definitions of Predicates

We explain coinductive definitions of $\mathbf{TID}_\nu$ and show some examples of formalization of streams by coinductive definitions.

Proposition 3.1.

Let ν be $\nu X.\lambda \mathbf{x}.A(X)$. Then

$$\forall \mathbf{x}(\nu(\mathbf{x}) \leftrightarrow A(\nu)) \qquad (\nu 1')$$

holds.

Proof 3.2.

By $(\nu 1)$, we get $\nu(\mathbf{x}) \to A(\nu)$. By letting C be $\lambda \mathbf{x}.A(\nu)$ in $(\nu 2)$, $A(\nu) \to \nu(\mathbf{x})$ holds. □

This proposition shows that $\nu P.\lambda \mathbf{x}.A(P)$ is the solution of the following recursive equation of a predicate P:

$$P(\mathbf{x}) \leftrightarrow A(P).$$

$(\nu 2)$ says that $\nu P.\lambda \mathbf{x}.A(P)$ is the greatest solution of this equation or the greatest fixed point of the function $\lambda P.\lambda x.A(P)$.

Streams can be formalized by coinductive definitions [3]. Therefore we can formalize streams in $\mathbf{TID}_\nu$.

We represent a stream by an infinite list $\langle a, s\rangle$ constructed by pairing where a is the first element of the stream, s is the rest of the stream. In this representation, if s is a stream, we can get the first element of s by s_0 and the rest by s_1.

We present an example of bit streams. A bit stream is a stream whose elements are 0 or 1. We will define a predicate $BS(x)$ which means that x is a bit stream. When we write down a formula $BS(x)$ in a naive way, BS itself occurs in the body of the definition as follows:

$$BS(x) \leftrightarrow x = \langle x_0, x_1\rangle \ \&\ (x_0 = 0 \lor x_0 = 1) \ \&\ BS(x_1).$$

BS is a solution P of the following equation for a predicate P

$$P(x) \leftrightarrow x = \langle x_0, x_1\rangle \ \&\ (x_0 = 0 \lor x_0 = 1) \ \&\ P(x_1) \qquad (1)$$

or the fixed point of the function

$$\lambda P.\lambda x.x = \langle x_0, x_1\rangle \ \&\ (x_0 = 0 \lor x_0 = 1) \ \&\ P(x_1). \ (2)$$

There may be many solutions P for (1). For example, $\lambda x.\bot$ is one solution of (1), though it is not our intended solution. $\lambda x.\bot$ is the least solution. Our intended solution is the greatest solution of (1) or the greatest fixed point of (2). Hence we have the solution in $\mathbf{TID}_\nu$ and it is represented as follows:

$$BS \equiv \nu P.\lambda x.x = \langle x_0, x_1\rangle \ \& \\ (x_0 = 0 \lor x_0 = 1) \ \&\ P(x_1).$$

Let $\overline{0}$ be $\mu s.\langle 0, s\rangle$. $\overline{0}$ represents the zero stream whose elements are all 0. We can show $BS(\overline{0})$ by coinduction $(\nu 2)$. Let C be $\lambda x.(x = \overline{0})$ in $(\nu 2)$, then we have

$$\forall x(x = \overline{0} \to \\ x = \langle x_0, x_1\rangle \ \&\ (x_0 = 0 \lor x_0 = 1) \ \&\ x_1 = \overline{0}) \\ \to \forall x(x = \overline{0} \to BS(x)).$$

By definition of $\overline{0}$,

$$\forall x(x = \overline{0} \to \\ x = \langle x_0, x_1\rangle \ \&\ (x_0 = 0 \lor x_0 = 1) \ \&\ x_1 = \overline{0})$$

holds and we have

$$\forall x(x = \overline{0} \to BS(x)).$$

Let $x = \overline{0}$ then we get $BS(\overline{0})$.

The coinductive definitions of predicates play an important role also to represent predicates of properties of streams [3, 6]. We will define the extensional equality $s \approx t$ for streams s and t. This equality can be represented by the coinductive definitions of predicates. $\approx$ is the greatest solution of the following equation for a predicate P:

$$P(x, y) \leftrightarrow x_0 = y_0 \ \&\ P(x_1, y_1).$$

Therefore $\approx$ can be formalized in $\mathbf{TID}_\nu$ as follows:

$$\approx \equiv \nu P.\lambda xy.x_0 = y_0 \ \&\ P(x_1, y_1).$$

4 Model of $\mathbf{TID}_\nu$

We will briefly explain semantics of $\mathbf{TID}_\nu$ by giving its intended model.

We will use classical set theory and the well-known greatest fixed point theorem for model construction in this section.

Theorem 4.1. (Greatest Fixed Point)

Suppose S be a set, $p(S)$ be a power set of S. If $f : p(S) \to p(S)$ is a monotone function, there exists a such that $a \in p(S)$ and

1. $f(a) = a$,

2. For any $b \in p(S)$, if $b \subset f(b)$, then $b \subset a$.

a is abbreviated as $\mathrm{gfp}(f)$.

We will construct a model M' of $\mathbf{TID}_\nu$ extending an arbitrary model M of $\mathbf{TID}^-$. Our intended model of $\mathbf{TID}^-$ is the closed total term model whose universe is the set of closed terms [1]. We denote the universe by U.

We will define $\rho \models A$ in almost the same way as for a first order logic where A is a formula and ρ is an environment which assigns a first order variable to an element of U and a predicate variable of arity n to a subset of U^n and which covers all the free first order variables and all the free predicate variables of A. We present only the definition for the case $(\nu P.\lambda \mathbf{x}.A(P))(\mathbf{t})$.

Define F as follows:

$$|\mathbf{x}| = n,$$
$$F : p(U^n) \to p(U^n),$$
$$F(X) = \{\mathbf{x} \in U^n \mid \rho[P := X] \models A(P)\},$$

where $\rho[P := X]$ is defined as follows:

$$\rho[P := X](P) = X,$$
$$\rho[P := X](x) = \rho(x) \qquad \text{if } x \text{ is not } P.$$

Then $\rho \models (\nu P.\lambda \mathbf{x}.A(P))(\mathbf{t})$ is defined as $\mathbf{t} \in \mathrm{gfp}(F)$. Note that F is monotone since a predicate variable P occurs only positively in $A(P)$.

Theorem 4.2.
If $\mathbf{TID}_\nu \vdash A$, then $\rho \models A$ for any environment ρ which covers all the free variables of A.

Theorem 4.3.
$\mathbf{TID}_\nu$ is consistent.

5 q-Realizability Interpretation of $\mathbf{TID}_\nu$

We will explain motivation of our realizability. We start with a usual q-realizability and try to interpret $(\nu P.\lambda x.A(P))(x)$. Let ν be $\nu P.\lambda x.A(P)$ and then $\nu(x) \leftrightarrow A(\nu, x)$ holds. We want to treat $\nu(x)$ and $A(\nu, x)$ in the same manner. So we require $(e \;\; \mathbf{q} \;\; \nu(x)) \leftrightarrow (e \;\; \mathbf{q} \;\; A(\nu, x))$. Therefore it is very natural to define $(e \;\; \mathbf{q} \;\; \nu(x))$ as $\nu^*(e, x)$ where $\nu^*(e, x)$ is the greatest solution of a recursive equation for a predicate variable X^*:

$$X^*(e, x) \leftrightarrow (e \;\; \mathbf{q} \;\; A(\nu, x))[(r \;\; \mathbf{q} \;\; \nu(y)) := X^*(r, y)].$$

where $[(r \;\; \mathbf{q} \;\; \nu(y)) := X^*(r, y)]$ of the right hand side means replacing each subformula $(r \;\; \mathbf{q} \;\; \nu(y))$ by a subformula $X^*(r, y)$ in a formula $(e \;\; \mathbf{q} \;\; A(\nu, x))$. We get the following definition of our realizability by describing syntactically this idea.

Our realizability in this paper is an extension of Grayson's realizability. We can also define usual q-realizability of coinductively defined predicates in the same way as in this paper.

Definition 5.1. (Harrop formula)

1. Atomic formulas $\bot$, $\mathbf{N}(a)$ and $a = b$ are Harrop.
2. If A and B are Harrop, then $A \,\&\, B$, $C \to B$, $\forall x A$ and $(\nu P.\lambda \mathbf{x}.A)(\mathbf{t})$ are also Harrop.

Since a Harrop formula does not have computational meanings, we can simplify the q-realizability interpretation of them.

Definition 5.2. (Abstract)

1. A predicate constant of arity n is an abstract of arity n.
2. A predicate variable of arity n is an abstract of arity n.
3. If A is a formula, $\lambda x_1 \ldots x_n.A$ is an abstract of arity n.

We identify $(\lambda x_1 \ldots x_n.A)(t_1, \ldots, t_n)$ with $A[x_1 := t_1, \ldots, x_n := t_n]$ where $[x_1 := t_1, \ldots, x_n := t_n]$ denotes a substitution.

Definition 5.3. (q-realizability Interpretation)
Suppose A is a formula, $P_1, \ldots, P_n$ is a sequence of predicate variables whose arities are $m_1, \ldots, m_n$ respectively and $F_1, G_1, \ldots, F_n, G_n$ is a sequence of abstracts whose arities are $m_1, m_1 + 1, \ldots, m_n, m_n + 1$ respectively.

$$(e \;\; \mathbf{q}_{P_1,\ldots,P_n}[F_1, G_1, \ldots, F_n, G_n] \;\; A)$$

is defined by induction on the construction of A as follows.

We abbreviate $\mathbf{q}_{P_1,\ldots,P_n}[F_1, G_1, \ldots, F_n, G_n]$ as $\mathbf{q}'$, $\mathbf{q}_{P_1,\ldots,P_n,P}[F_1, G_1, \ldots, F_n, G_n, F, G]$ as $\mathbf{q}'_P[F, G]$, $F_1, \ldots, F_n$ as $\mathbf{F}$ and $P_1, \ldots, P_n$ as $\mathbf{P}$.

1. $(e \;\; \mathbf{q}' \;\; A) \equiv e = 0 \,\&\, A_{\mathbf{P}}[\mathbf{F}]$ where A is Harrop.
2. $(e \;\; \mathbf{q}' \;\; P_i(\mathbf{t})) \equiv F_i(\mathbf{t}) \,\&\, G_i(e, \mathbf{t})$.
3. $(e \;\; \mathbf{q}' \;\; Q(\mathbf{t})) \equiv Q(\mathbf{t}) \,\&\, Q^*(e, \mathbf{t})$ where $Q \not\equiv P_i$ $(1 \le i \le n)$.
4. $(e \;\; \mathbf{q}' \;\; A \,\&\, B) \equiv (e_0 \;\; \mathbf{q}' \;\; A) \,\&\, (e_1 \;\; \mathbf{q}' \;\; B)$.
5. $(e \;\; \mathbf{q}' \;\; A \vee B) \equiv \mathbf{N}(e_0) \,\&$
   $(e_0 = 0 \to (e_1 \;\; \mathbf{q}' \;\; A)) \,\&$
   $(e_0 \neq 0 \to (e_1 \;\; \mathbf{q}' \;\; B))$.
6. $(e \;\; \mathbf{q}' \;\; A \to B) \equiv (A \to B)_{\mathbf{P}}[\mathbf{F}] \,\&\, \forall q((q \;\; \mathbf{q}' \;\; A) \to (eq \;\; \mathbf{q}' \;\; B))$.
7. $(e \;\; \mathbf{q}' \;\; \forall x A(x)) \equiv \forall x(ex \;\; \mathbf{q}' \;\; A(x))$.
8. $(e \;\; \mathbf{q}' \;\; \exists x A(x)) \equiv (e_1 \;\; \mathbf{q}' \;\; A(e_0))$.
9. $(e \;\; \mathbf{q}' \;\; (\nu X.\lambda \mathbf{x}.A(X))(\mathbf{t})) \equiv$
   $(\nu X^*.\lambda e\mathbf{x}.(e \;\; \mathbf{q}'_X[\nu_{\mathbf{P}}[\mathbf{F}], X^*] \;\; A(X)))(e, \mathbf{t})$
   where $\nu \equiv \nu X.\lambda \mathbf{x}.A(X)$.

In the above definition, ${}_{P_1,\ldots,P_n}[F_1, G_1, \ldots, F_n, G_n]$ means a substitution. Our realizability interpretation is something like a realizability interpretation with a substitution.

Proposition 5.4.
Let $\nu \equiv \nu P.\lambda \mathrm{x}.A(P)$.

1. $\forall \mathrm{x} r((r \ \ \mathbf{q} \ \ \nu(\mathrm{x})) \leftrightarrow (r \ \ \mathbf{q} \ \ A(\nu)))$.

2. $\lambda \mathrm{x} r.r \ \ \mathbf{q} \ \ \forall \mathrm{x}(\nu(\mathrm{x}) \rightarrow A(\nu))$.

Proof 5.5.
By the definition of q-realizability and $(\nu 1')$. □

Definition 5.6.
For a formula A, a predicate variable P and a term f, we define a term $\sigma_A^{P,f}$ by induction on the construction of A as follows:

1. A is a Harrop formula, then $\sigma_A^{P,f} \equiv \lambda r.0$.

2. $A \equiv P(\mathbf{t})$, then $\sigma_A^{P,f} \equiv \lambda r.f\mathbf{t}r$.

3. $A \equiv Q(\mathbf{t})$, then $\sigma_A^{P,f} \equiv \lambda r.r$ if $Q \not\equiv P$.

4. $A \equiv A_1 \,\&\, A_2$, then $\sigma_A^{P,f} \equiv \lambda r.\langle \sigma_{A_1}^{P,f} r_0, \sigma_{A_2}^{P,f} r_1 \rangle$.

5. $A \equiv A_1 \vee A_2$, then $\sigma_A^{P,f} \equiv \lambda r.\langle r_0, \mathrm{d} r_0 0 \sigma_{A_1}^{P,f} \sigma_{A_2}^{P,f} r_1 \rangle$.

6. $A \equiv A_1 \rightarrow A_2$, then $\sigma_A^{P,f} \equiv \lambda r q.\sigma_{A_2}^{P,f}(r(\sigma_{A_1}^{P,f} q))$.

7. $A \equiv \forall x A_1(x)$, then $\sigma_A^{P,f} \equiv \lambda r x.\sigma_{A_1(x)}^{P,f}(rx)$.

8. $A \equiv \exists x A_1(x)$, then $\sigma_A^{P,f} \equiv \lambda r.\langle r_0, \sigma_{A_1(r_0)}^{P,f} r_1 \rangle$.

9. $A \equiv (\nu Q.\lambda \mathrm{x}.A_1)(\mathbf{t})$, then $\sigma_A^{P,f} \equiv (\mu g.\lambda \mathrm{x} r.\sigma_{A_1}^{Q,g}(\sigma_{A_1}^{P,f} r))\mathbf{t}$ where $Q \not\equiv P$.

Proposition 5.7.
Let $\nu \equiv \nu P.\lambda \mathrm{x}.A(P)$. Then

$$\lambda q.\mu f.\lambda \mathrm{x} r.\sigma_{A(P)}^{P,f}(q\mathrm{x}r) \ \ \mathbf{q} \ \ \forall \mathrm{x}(C(\mathrm{x}) \rightarrow A(C)) \rightarrow \forall \mathrm{x}(C(\mathrm{x}) \rightarrow \nu(\mathrm{x}))$$

holds.

We prove it in Appendix B.

Theorem 5.8. (Soundness Theorem)
If $\mathbf{TID}_\nu \vdash A$, we can get a term e from the proof of $\vdash A$ and $\mathbf{TID}_\nu \vdash (e \ \ \mathbf{q} \ \ A)$ holds where all the free variables of e are included in all the free variables of A.

Proof 5.9.
By induction on the proof of $\vdash A$. The case of the axiom $(\nu 1)$ is proved by Proposition 5.4. The case of the axiom $(\nu 2)$ is proved by Proposition 5.7. □

6 Program Synthesis with Streams

In this section, we give general program extraction theorem, stream program extraction theorem for $\mathbf{TID}_\nu$ and an example of program synthesis.

Program synthesis by theorem proving techniques has been studied both in typed theories [2] and untyped theories [4]. For untyped theories, realizability interpretation is used as the foundation of program synthesis by theorem proving techniques. In Section 3, we showed that streams and programs which treat streams can be formalized in $\mathbf{TID}_\nu$ by the facility of coinductively definitions of predicates. In Section 5, we showed that realizability interpretation can be defined for $\mathbf{TID}_\nu$ and the interpretation is sound. Hence we can synthesize programs which treat streams by theorem proving techniques in $\mathbf{TID}_\nu$ using realizability interpretation.

We represent streams by infinite lists constructed by pairing. We represent a specification of a program by a formula:

$$\forall x(A(x) \rightarrow \exists y B(x,y))$$

where x is an input, y is an output, $A(x)$ is an input condition and $B(x,y)$ is an input output relation.

Theorem 6.1. (Program Extraction)
Suppose that we prove a specification formula $\forall x(A(x) \rightarrow \exists y B(x,y))$ of a program in $\mathbf{TID}_\nu$ and we have a realizer j such that

$$\forall x(A(x) \rightarrow (jx \ \ \mathbf{q} \ \ A(x))).$$

Then we can get a program f and a proof of

$$\forall x(A(x) \rightarrow B(x, fx))$$

effectively from the proof of the specification formula.

Proof 6.2.
Since the specification formula is proved in $\mathbf{TID}_\nu$, by soundness theorem of q-realizability interpretation we have a realizer e such that

$$e \ \ \mathbf{q} \ \ \forall x(A(x) \rightarrow \exists y B(x,y))$$

holds. Let f be $\lambda x.(ex(jx))_0$. Then the claim holds. □

We can synthesize programs in the following steps:

1. We write down a specification formula.

2. We prove the specification formula in $\mathbf{TID}_\nu$.

3. We extract a program from the proof.

The program extraction theorem says that the third step can be automated completely.

Example 6.3.
We show an example of the program which gets a stream of natural numbers and returns a stream whose each element is the element of the input stream plus one.

The predicate $NS(x)$ which says that x is a stream of natural numbers can be represented in $\mathbf{TID}_\nu$ by the facility of coinductive definitions of predicates as follows:

$NS \equiv \nu X.\lambda x.x = \langle x_0, x_1\rangle \;\&\; \mathrm{N}(x_0) \;\&\; X(x_1)$.

The input condition of the specification is a formula $NS(x)$.

The input output relation of the specification is a formula $ADD1(x, y)$ which is defined as follows:

$ADD1 \equiv \nu X.\lambda xy.y_0 = x_0 + 1 \;\&\; X(x_1, y_1)$.

The specification formula is:

$\forall x(NS(x) \to \exists y ADD1(x, y))$.

We have one problem for this program synthesis method. The coinduction cannot be applied to the part $\forall x(NS(x) \to \ldots)$ in the above example. We cannot prove $\exists y ADD1(x, y)$ by the coinduction in general. Therefore the realizer of the coinduction cannot give a loop structure for the program. On the other hand, a realizer of the induction principle plays an important role for this approach of program synthesis since the realizer corresponds to a loop structure of a program [4, 7]. Therefore we need the new method by which a realizer of the coinduction also corresponds to a loop structure and is useful.

Then we need more specialized program extraction method for programs with streams in which the coinduction is useful. We give one solution for this problem by the next theorem.

We put 2 restrictions on the theorem: One is that the input condition $A(x)$ must be the form $(\nu X.\lambda x.x = \langle x_0, x_1\rangle \;\&\; \tilde{A}(x_0) \;\&\; X(x_1))(x)$ for some $\tilde{A}$. The other is that the input output relation $B(x, y)$ must be the form $(\nu X.\lambda xy.\widetilde{B}(x, y_0) \;\&\; X(x_1, y_1))(x, y)$ for some $\widetilde{B}$. These restrictions require an input condition and an input output relation are uniform over data and they are natural when we suppose that an input x and an output y are both streams.

Theorem 6.4. (Stream Program Extraction)
Suppose that the specification formula is $\forall x(A(x) \to \exists y B(x, y))$,

$A \equiv \nu X.\lambda x.x = \langle x_0, x_1\rangle \;\&\; \tilde{A}(x_0) \;\&\; X(x_1)$,

$B \equiv \nu X.\lambda xy.\widetilde{B}(x, y_0) \;\&\; X(x_1, y_1)$

and we have a term j such that $\forall x(A(x) \to (jx \;\; \mathbf{q} \;\; A(x)))$. Then we define

$B^\circ \equiv \nu X.\lambda x.\exists z \widetilde{B}(x, z) \;\&\; X(x_1)$.

If we have e such that

$e \;\; \mathbf{q} \;\; \forall x(A(x) \to B^\circ(x))$,

we can get a term F such that

$\forall x(A(x) \to B(x, Fx))$

where

$\mathit{filter} \equiv \mu f.\lambda x.\langle x_{00}, f x_1\rangle$,

$F \equiv \lambda x.\mathit{filter}(ex(jx))$.

We prove it in Appendix C.

By this theorem, we can synthesize programs in the following steps:

1. We write down a specification formula $\forall x(A(x) \to \exists y B(x, y))$.
2. We prove the corresponding formula $\forall x(A(x) \to B^\circ(x))$ in $\mathbf{TID}_\nu$.
3. We extract a program $\lambda x.\mathit{filter}(ex(jx))$ from the proof where e is a realizer of the corresponding formula $\forall x(A(x) \to B^\circ(x))$.

In the second step, we can apply the coinduction to prove the part $B^\circ(x)$ since $B^\circ(x)$ is defined by coinductive definitions. Therefore a realizer of the coinduction can correspond to a loop structure of the program.

Example 6.5.
We treat the same example as above again. The specification formula is a formula $\forall x(NS(x) \to \exists y ADD1(x, y))$. Hence the formula $ADD1^\circ(x)$ is:

$$ADD1^\circ \equiv \nu X.\lambda x.\exists z(z = x_0 + 1) \;\&\; X(x_1). \qquad (3)$$

Therefore the corresponding formula we must prove is:

$$\forall x(NS(x) \to ADD1^\circ(x)). \qquad (4)$$

If we prove this formula in $\mathbf{TID}_\nu$, we can get the program which satisfies the specification by stream program extraction theorem.

The conditions of the theorem hold for this case. We can put $j \equiv \lambda x.\mu s.\langle 0, s\rangle$ since

$\forall x(NS(x) \to (\mu s.\langle 0, s\rangle \;\; \mathbf{q} \;\; NS(x)))$.

We prove (4) in the following way here: Firstly, we prove

$$\forall x(NS(x) \to \exists z(z = x_0 + 1) \;\&\; NS(x_1)). \qquad (5)$$

This is proved by letting z be $x_0 + 1$. Secondly, by letting C be NS in $(\nu 2)$ for $ADD1^\circ$, we have

$$\forall x(NS(x) \to \exists z(z = x_0 + 1) \;\&\; NS(x_1)) \to$$
$$\forall x(NS(x) \to ADD1^\circ(x)). \qquad (6)$$

Finally, by (5) and (6), we get (4).

We calculate realizers corresponding to the above proofs as follows: The realizer corresponding to the proof of (5) is:

$e_1 \equiv \lambda xr.\langle\langle x_0 + 1, 0\rangle, r_{11}\rangle$,

$e_1 \;\; \mathbf{q} \;\; \forall x(NS(x) \to \exists z(z = x_0 + 1) \;\&\; NS(x_1))$.

The realizer corresponding to the proof of (6) is:

$e_2 \equiv \lambda q.\mu f.\lambda xr.\sigma(qxr)$,

$e_2 \;\; \mathbf{q} \;\; \forall x(NS(x) \to \exists z(z = x_0 + 1) \;\&\; NS(x_1)) \to$
$\forall x(NS(x) \to ADD1^\circ(x))$

where

$\sigma \equiv \lambda r.\langle\langle r_{00}, r_{01}\rangle, f x_1 r_1\rangle$.

The realizer corresponding to the proof of (4) is:

$e \equiv e_2 e_1$,

$e \;\; \mathbf{q} \;\; \forall x(NS(x) \to ADD1^\circ(x))$.

We get

$$e = \mu f.\lambda xr.\langle\langle x_0 + 1, 0\rangle, f x_1 r_{11}\rangle.$$

The extracted program is:

$$\begin{aligned} Fx &= \mathit{filter}(ex(jx)) \\ &= \mathit{filter}(fx(\mu s.\langle 0, s\rangle)) \\ &= (\mu g.\lambda x.\langle x_0 + 1, g x_1\rangle)x \end{aligned}$$

where $f \equiv \mu f.\lambda xr.\langle\langle x_0 + 1, 0\rangle, f x_1 r_{11}\rangle$. This is the program we expect.

Remark that the realizer e_2 of the coinduction (6) gives a loop structure of the program F.

Acknowledgments

I would like to thank Mr. Satoshi Kobayashi and Mr. Yukiyoshi Kameyama for careful comments. I'm deeply grateful to Prof. Masahiko Sato for invaluable discussions and comments.

References

[1] M. Beeson, *Foundations of Constructive Mathematics* (Springer, 1985).

[2] R.L. Constable et al., *Implementing Mathematics with the Nuprl Proof Development System* (Prentice-Hall, 1986).

[3] P. Dybjer and H.P. Sander, A Functional Programming Approach to the Specification and Verification of Concurrent Systems, *Formal Aspects of Computing* 1 (1989) 303–319.

[4] S. Hayashi and H. Nakano, **PX***: A Computational Logic* (MIT Press, Cambridge, 1988).

[5] S. Kobayashi, Inductive/Coinductive Definitions and Their Realizability Interpretation, Manuscript (1991).

[6] R. Milner, *Communication and Concurrency* (Prentice Hall, 1989).

[7] M. Tatsuta, Program Synthesis Using Realizability, *Theoretical Computer Science* **90** (1991) 309–353.

[8] M. Tatsuta, Realizability Interpretation of Greatest Fixed Points, Manuscript (1991).

[9] M. Tatsuta, Monotone Recursive Definition of Predicates and Its Realizability Interpretation, *Proceedings of Theoretical Aspects of Computer Software,* LNCS **526** (1991) 38–52.

Appendix

A Axioms and Inference Rules of TID_ν

The logical axioms and inference rules are the same as the ones of a usual intuitionistic logic.

Axioms for Equality:

$$\forall x(x = x) \tag{E1}$$

$$\forall x, y(x = y \,\&\, A(x) \to A(y)) \tag{E2}$$

Axioms for Combinators:

$$\forall x, y(\mathbf{K}xy = x) \tag{C1}$$

$$\forall x, y, z(\mathbf{S}xyz = xz(yz)) \tag{C2}$$

Axioms for Pairing:

$$\forall x, y(\mathrm{p}_0(\mathrm{p}xy) = x) \tag{P1}$$

$$\forall x, y(\mathrm{p}_1(\mathrm{p}xy) = y) \tag{P2}$$

Axioms for Natural Numbers:

$$\mathrm{N}(0) \tag{N1}$$

$$\forall x(\mathrm{N}(x) \to \mathrm{N}(\mathrm{s_N}x)) \tag{N2}$$

$$\forall x(\mathrm{N}(x) \to \mathrm{p_N}(\mathrm{s_N}x) = x) \tag{N3}$$

$$\forall x(\mathrm{N}(x) \to \mathrm{s_N}x \neq 0) \tag{N4}$$

$$A(0) \,\&\, \forall x(\mathrm{N}(x) \,\&\, A(x) \to A(\mathrm{s_N}x)) \to \forall x(\mathrm{N}(x) \to A(x)) \tag{N5}$$

Axioms for d:

$$\forall x, y, a, b(\mathrm{N}(x) \,\&\, \mathrm{N}(y) \,\&\, x = y \to \mathrm{d}xyab = a) \tag{D1}$$

$$\forall x, y, a, b(\mathrm{N}(x) \,\&\, \mathrm{N}(y) \,\&\, x \neq y \to \mathrm{d}xyab = b) \tag{D2}$$

B Proof of Soundness Theorem

Lemma B.1.

(1) If a predicate variable P occurs only positively in a formula A,

$$(r \;\; \mathbf{q}_P[F, \lambda y\mathrm{x}.(y \;\; \mathbf{q} \;\; C(\mathrm{x}))] \;\; A) \to (\sigma_A^{P,f} r \;\; \mathbf{q}_P[F, \lambda y\mathrm{x}.\exists r((r \;\; \mathbf{q} \;\; C(\mathrm{x})) \,\&\, y = f\mathrm{x}r)] \;\; A).$$

(2) If a predicate variable P occurs only negatively in a formula A,

$$(r \;\; \mathbf{q}_P[F, \lambda y\mathrm{x}.\exists r((r \;\; \mathbf{q} \;\; C(\mathrm{x})) \,\&\, y = f\mathrm{x}r)] \;\; A) \to (\sigma_A^{P,f} r \;\; \mathbf{q}_P[F, \lambda y\mathrm{x}.(y \;\; \mathbf{q} \;\; C(\mathrm{x}))] \;\; A).$$

Proof B.2.

We prove (1) and (2) simultaneously by induction on the construction of A. □

Proof B.3. (of 5.7)

Let $\nu \equiv \nu P.\lambda\mathrm{x}.A(P)$.

Suppose

$$\forall \mathrm{x}(C(\mathrm{x}) \to A(C)),$$

$$q \;\; \mathbf{q} \;\; \forall \mathrm{x}(C(\mathrm{x}) \to A(C))$$

and let

$$f \equiv \mu f.\lambda \mathrm{x}r.\sigma_{A(P)}^{P,f}(q\mathrm{x}r).$$

We show

$$f \;\; \mathbf{q} \;\; \forall \mathrm{x}(C(\mathrm{x}) \to \nu(\mathrm{x})).$$

Let $\nu^*(r,\mathrm{x}) \equiv (r \ \mathbf{q}\ \ \nu(\mathrm{x}))$. It is sufficient to show

$$\forall \mathrm{x}r((r \ \ \mathbf{q}\ \ C(\mathrm{x})) \to \nu^*(f\mathrm{x}r,\mathrm{x})).$$

This is equivalent to

$$\forall \mathrm{x}y(\exists r((r \ \ \mathbf{q}\ \ C(\mathrm{x})) \,\&\, y = f\mathrm{x}r) \to \nu^*(y,\mathrm{x})).$$

By (ν2), it is sufficient to show

$$\forall \mathrm{x}y(\exists r((r \ \ \mathbf{q}\ \ C(\mathrm{x})) \,\&\, y = f\mathrm{x}r) \to$$
$$(y \ \ \mathbf{q}_P[\nu, \lambda y\mathrm{x}.\exists r((r \ \ \mathbf{q}\ \ C(\mathrm{x})) \,\&\, y = f\mathrm{x}r)] \ \ A(P))).$$

This is equivalent to

$$\forall \mathrm{x}r((r \ \ \mathbf{q}\ \ C(\mathrm{x})) \to$$
$$(f\mathrm{x}r \ \ \mathbf{q}_P[\nu, \lambda y\mathrm{x}.\exists r((r \ \ \mathbf{q}\ \ C(\mathrm{x})) \,\&\, y = f\mathrm{x}r)]$$
$$A(P))).$$

Fix x and r and assume

$$r \ \ \mathbf{q}\ \ C(\mathrm{x}).$$

We show

$$f\mathrm{x}r \ \ \mathbf{q}_P[\nu, \lambda y\mathrm{x}.\exists r((r \ \ \mathbf{q}\ \ C(\mathrm{x})) \,\&\, y = f\mathrm{x}r)] \ \ A(P).$$

By the assumption about q,

$$q\mathrm{x}r \ \ \mathbf{q}\ \ A(C).$$

Hence

$$q\mathrm{x}r \ \ \mathbf{q}_P[C, \lambda y\mathrm{x}.(y \ \ \mathbf{q}\ \ C(\mathrm{x}))] \ \ A(P).$$

By positivity and $\forall \mathrm{x}(C(\mathrm{x}) \to \nu(\mathrm{x}))$,

$$q\mathrm{x}r \ \ \mathbf{q}_P[\nu, \lambda y\mathrm{x}.(y \ \ \mathbf{q}\ \ C(\mathrm{x}))] \ \ A(P).$$

By Lemma B.1,

$$\sigma^{P,f}_{A(P)}(q\mathrm{x}r) \ \ \mathbf{q}_P[\nu, \lambda y\mathrm{x}.\exists r((r \ \ \mathbf{q}\ \ C(\mathrm{x})) \,\&\, y = f\mathrm{x}r)]$$
$$A(P).$$

By $f\mathrm{x}r = \sigma^{P,f}_{A(P)}(q\mathrm{x}r)$, we have

$$f\mathrm{x}r \ \ \mathbf{q}_P[\nu, \lambda y\mathrm{x}.\exists r((r \ \ \mathbf{q}\ \ C(\mathrm{x})) \,\&\, y = f\mathrm{x}r)] \ \ A(P).$$

□

C Proof of Stream Extraction Theorem

Lemma C.1.

Suppose that

$$A \equiv \nu X.\lambda x.x = \langle x_0, x_1\rangle \,\&\, \tilde{A}(x_0) \,\&\, X(x_1),$$
$$B \equiv \nu X.\lambda xy.\tilde{B}(x, y_0) \,\&\, X(x_1, y_1),$$
$$B^\circ \equiv \nu X.\lambda x.\exists z\tilde{B}(x,z) \,\&\, X(x_1).$$

Then

$$\forall f(\forall x(A(x) \to (fx \ \ \mathbf{q}\ \ B^\circ(x))) \to$$
$$\forall x(A(x) \to B(x, \mathit{filter}(fx))))$$

holds.

Proof C.2.

By only rules of **NJ**, the above goal is equivalent to

$$\forall xy(\exists f(\forall x(A(x) \to (fx \ \ \mathbf{q}\ \ B^\circ(x))) \,\&$$
$$A(x) \,\&\, y = \mathit{filter}(fx)) \to B(x,y)).$$

By (ν2), it is sufficient to show

$$\forall xy(\exists f(\forall x(A(x) \to (fx \ \ \mathbf{q}\ \ B^\circ(x))) \,\&$$
$$A(x) \,\&\, y = \mathit{filter}(fx)) \to$$
$$\tilde{B}(x, y_0) \,\&\, \exists g(\forall x(A(x) \to (gx \ \ \mathbf{q}\ \ B^\circ(x))) \,\&$$
$$A(x_1) \,\&\, y_1 = \mathit{filter}(gx_1))).$$

By only rules of **NJ**, it is equivalent to

$$\forall xf(\forall x(A(x) \to (fx \ \ \mathbf{q}\ \ B^\circ(x)) \,\&\, A(x) \to$$
$$\tilde{B}(x, (\mathit{filter}(fx))_0) \,\&$$
$$\exists g(\forall x(A(x) \to (gx \ \ \mathbf{q}\ \ B^\circ(x))) \,\&$$
$$A(x_1) \,\&\, (\mathit{filter}(fx))_1 = \mathit{filter}(gx_1))). \tag{7}$$

We will prove it.

Fix x and f and assume that

$$\forall x(A(x) \to (fx \ \ \mathbf{q}\ \ B^\circ(x)), \tag{8}$$
$$A(x). \tag{9}$$

By (8) and (9), $(fx \ \ \mathbf{q}\ \ B^\circ(x))$ holds. Hence

$$((fx)_{01} \ \ \mathbf{q}\ \ \tilde{B}(x,(fx)_{00})) \,\&\, ((fx)_1 \ \ \mathbf{q}\ \ B^\circ(x_1)) \tag{10}$$

holds. Therefore $\tilde{B}(x, (\mathit{filter}(fx))_0)$ holds since $(\mathit{filter}(fx))_0 = (fx)_{00}$.

Put g be $\lambda y.(f\langle x_0, y\rangle)_1$. We will show $\forall y(A(y) \to (gy \ \ \mathbf{q}\ \ B^\circ(y)))$. Fix y and assume that $A(y)$. By the definition of A,

$$A(x) \leftrightarrow x = \langle x_0, x_1\rangle \,\&\, \tilde{A}(x_0) \,\&\, A(x_1)$$

and

$$A(\langle x_0, y\rangle) \leftrightarrow \tilde{A}(x_0) \,\&\, A(y)$$

hold. By this and (9), $\tilde{A}(x_0)$ holds. Hence $A(\langle x_0, y\rangle)$ holds. Combined it with (8), we get $(f\langle x_0,y\rangle \ \ \mathbf{q}\ \ B^\circ(\langle x_0, y\rangle))$. Hence $((f\langle x_0,y\rangle)_1 \ \ \mathbf{q}\ \ B^\circ(y))$ and $(gy \ \ \mathbf{q}\ \ B^\circ(y))$ hold. Therefore we get $\forall y(A(x) \to (gy \ \ \mathbf{q}\ \ B^\circ(y)))$.

By (9), $A(x_1)$ holds. Since, in general, $(\mathit{filter}(s))_1 = \mathit{filter}(s_1)$ holds, we get $(\mathit{filter}(fx))_1 = \mathit{filter}((fx)_1) = \mathit{filter}(gx_1)$. Therefore (7) holds. □

Proof C.3. (of Theorem 6.4)

By the assumptions and the definition of q-realizability, $\forall x(A(x) \to (ex(jx) \ \ \mathbf{q}\ \ B^\circ(x)))$ holds. Letting f be $\lambda x.ex(jx)$ in Lemma C.1, we get $\forall x(A(x) \to B(x, Fx))$. □