ICOT Technical Report: TR-573

TR-573

Extended Projection Method and Realizability Interpretation

by
Y. Takayama & S. Hayashi (Oki)

July, 1990

©1990, ICOT

ICOT

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

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

To be submitted for publication in *Information and Computation*

EXTENDED PROJECTION METHOD AND REALIZABILITY INTERPRETATION (June 5, 1990)

Yukihide Takayama* and Susumu Hayashi**

*) Electrical System Laboratory, OKI Electric Industry Co., Ltd.
11-22 Sibaura 4 chome, Minato-ku, Tokyo 108, Japan
takayama@okilab.oki.co.jp

**) Department of Applied Mathematical and Informatics, Ryukoku University,
Seta, Otsu, Siga, 520-21, Japan
hayashi@rins.ryukoku.ac.jp

ABSTRACT

A new application of the Kreisel-Troelstra realizability interpretation is given in this paper. The realizability was originally introduced to give a semantics of an intuitionistic second order number theory, and has been used to give a semantics of polymorphism and type theoretic treatment of module structure. This paper investigates first order Kreisel-Troelstra realizability to generate redundancy free programs from constructive proofs, and show that the realizability gives a good theoretical background for program extraction.
Keywords: constructive logic, realizability, program extraction, proof transformation

1. Introduction

Constructive logics or constructive type theories can be used to describe specifications of programs as proofs of theorems, to check the correctness of them and to extract programs from the proofs. For the program extraction from constructive proofs, Curry-Howard's correspondence of types and formulas (Howard, 1980) or realizability interpretations (see, for example, Troelstra, 1973) are used.

One of the main problems in the program extraction is how to extract redundancy-free codes from proofs. For example, the standard interpretation of the formula $\exists x.A$ in intuitionistic logic is a pair of a value, t, of x and a justification of $A_x[t]$. However, only the value of x is needed as the extracted program in many cases. Several techniques have been developed to refrain from generating the justification of $A_x[t]$, i.e., the redundant code, such as subset

types (Nordström & Petersson, 1981; Constable, 1986), rank 0 formulas (Hayashi & Nakano, 1988; type 0 formulas in Hayashi, 1986). Their idea is to introduce new logical constants. Also, the Calculus of Constructions with two constants, *Prop* and *Spec* (Paulin-Mohring, 1989) has two kinds of formulas: formulas from which computational contents should and should not be extracted. Another technique called the extended projection method has been developed by one of the authors (Takayama, 1989; Takayama 1990). The idea is to introduce two kinds of logical constants: logical constants with constructive interpretation and non-constructive interpretation. For example, constructive interpretation of $\exists x$ in $\exists x.A$ is the value of x, but from the formula $\exists' x.A$ with checked $\exists$, $\exists'$, the value of x is not extracted. The disjunction, $\vee$, is handled similarly. This idea allows fine grained specification of redundancy. For example, it is impossible to specify in rank 0 formulas that only the value of y is unnecessary in $\exists x.\exists y.\exists z.A(x,y,z)$. Another advantage of this idea is that it is rather easy to give an algorithm of semi-automatic analysis of redundancy ($Mark$). Given a proof of $\exists x.\exists y.\exists z.A(x,y,z)$, if the formula is changed to a formula with checked logical constants, $\exists x.\exists' y.\exists z.A'(x,y,z)$ ($A'(x,y,z)$ is a formula with some logical constants checked in $A(x,y,z)$), the algorithm translates the original proof to a new proof with the checked logical constants, called a marked proof, from which a redundancy free program can be extracted. Therefore, the method has the following pleasing property with proof normalization and execution of extracted programs:

$$\begin{array}{ccccc} Proof_0 & \overset{normalization}{\longrightarrow} & Proof_1 & \overset{marking}{\longrightarrow} & Marked\ proof \\ \downarrow Ext & & \downarrow Ext & & \downarrow NExt \\ Code_0 & \overset{\beta-reduction}{\longrightarrow} & Code_1 & \overset{projection}{\longrightarrow} & Code_2 \end{array}$$

where Ext and $NExt$ are the program extraction procedure.
This means that execution of programs or program transformation can be described as proof transformation. However, the proof transformation, $Mark$, from proofs to marked proofs is not defined in a constructive logic. A marked proof is actually a proof tree with annotation, but the tree itself is not a proof tree. Although Ext performs a realizability interpretation, $NExt$ is given at a meta-logical level and does not correspond to any realizability interpretation.
This paper shows that a constructive logic with a new logical constant, which is interpreted by Kreisel-Troelstra realizability (Kreisel & Troelstra, 1979), can be used to describe the program transformation from proofs to marked proofs and the program extraction from marked proofs. In other words, all part of the following commutative diagram can be described uniformly in the new constructive logic:

$$\begin{array}{ccccc} Proof_0 & \overset{normalization}{\longrightarrow} & Proof_1 & \overset{Tr}{\longrightarrow} & Proof_2 \\ \downarrow Ext & & \downarrow Ext & & \downarrow Ext \\ Code_0 & \overset{\beta-reduction}{\longrightarrow} & Code_1 & \overset{projection}{\longrightarrow} & Code_2 \end{array}$$

where Tr and $Proof_2$ correspond to $Mark$ and $Marked\ proof$ in the previous diagram, and there is no longer a need to prepare two kinds of program extraction.
The structure of this paper is as follows. Section 2 gives an outline of the extended projection method. Section 3 defines QPC^{KT} which is a first order constructive logic with new existential quantifier, $\tilde{\exists}$. The realizability interpretation of QPC^{KT} is defined in section 4. The quantifier, $\tilde{\exists}$, is interpreted by first order Kreisel-Troelstra realizability. The soundness theorem is also proved. The program extraction algorithm based on the realizability interpretation is given in section 5. The proof transformation rule, Tr, which corresponds to the $Mark$ procedure in the original extended projection method is defined in section 6. The relation with the original extended projection method is formally given in section 7. Examples are investigated in section 8. Comparison with other works is explained in section 9, and section 10 is a concluding remark and the future work.

2. Overview of the Original Extended Projection Method

2.1 QPC and its realizability

The original extended projection method, (Takayama, 89; Takayama, 90), is given to a first order constructive logic called QPC (Takayama, 1988). One of the unique features of QPC resides in its realizability interpretation which is a variant of **q**-realizability (Troelstra, 1973; Beeson, 1985). The realizer of A is, or equal to, a sequence of terms:

$$(t_1, \cdots, t_n)\ \mathbf{q}\ A$$

and each element, t_i, of the sequence corresponds to the disjunction symbol or the existential quantifiers which occur in the strictly positive part of A. For example, assume that A is $\exists x.1 \leq x \leq 3 \supset (\exists y.x = y \vee \exists z.x = z + 1)$. Then, the realizability of A is in the following form:

$$(t_1, t_2, t_3)\ \mathbf{q}\ \exists x.1 \leq x \leq 3 \supset (\exists y.x = y \vee \exists z.x = z + 1)$$

t_1 corresponds to $\vee$ and it is the information indicating which one holds in the formulas connected by $\vee$. t_2 and t_3 correspond to $\exists y$ and $\exists z$, and are values of y and z. Note that t_i is a function that takes a realizer of $\exists x.1 \leq x \leq 3$ as input.
The essential difference from the standard **q**-realizability is the interpretation of atomic formulas. Any term can be a realizer of an atomic formula in the standard **q**-realizability, but the realizer is restricted to nil sequence, (), in QPC.

2.2 Marking System and Mark Procedure

The realizability induces the notion of length of formulas: If A is a formula, then the length of A, $l(A)$, is the number of $\vee$ and $\exists$ in the strictly positive part of A, and at the same time it is equal to the length of the realizer of A as a sequence of terms. Any part of the realizer of A can be indicated by the position numbers: $1, 2, \cdots, l(A)$. Therefore, the powerset $\mathbf{P}(\{1, 2, \cdots, l(A)\})$ can be used to specify which part of the realizer should be extracted as a program. An element of $\mathbf{P}(\{1, 2, \cdots, l(A)\})$ given to A is a *marking* of A. In particular, a marking given to the end formula of a given proof is called a *declaration*. If a declaration is given to a proof tree, it can be propagated to each node of the tree from the end formula to top formulas according to the inference rules used at each node. This procedure is called the marking procedure, $Mark$. The tree made by $Mark$ is called a *marked proof tree*.

2.3 $NExt$ procedure

The program extraction from the marked proof trees is performed by the $NExt$ procedure. The procedure is obtained by modifying the original program extraction procedure Ext, and it extracts the part of the realizer of the end formula specified by the declaration. The advantage of the extended projection method besides the fine grained specification of redundancy is that it allows extraction of multiple programs from a proof. The marking procedure only attaches additional information to every node in the given proof tree, but does not change the proof itself. Therefore, it is easy to extract multiple programs from a given proof by changing the declaration.

2.4 Marking Condition

The marking procedure can be performed mechanically, but it does not always succeed for proofs in induction. Assume that the following proof in mathematical induction, and that a declaration, S, is given:

$$\dfrac{\begin{array}{cc} & [A(x-1)] \\ \Sigma_1 & \Sigma_2 \\ A(x) & A(x) \end{array}}{\{\forall x.A(x)\}_S}(ind)$$

The marking, S, is propagated to the occurrences of induction hypothesis, $A(x-1)$. Let T be the union of the markings attached to all the occurrences of $A(x-1)$. Then, if $T \not\subseteq S$, $NExt$ generates, for example, the following incomplete mutually defined functions:

$$F(X) \leftarrow \xi(X, F, G, H)$$

$$G(X) \leftarrow \zeta(X, F, G, H)$$

F and G are indicated by S, while T indicates F, G and H. No definition of H is generated in this case. Consequently, this case is regarded as a failure of the marking procedure. The

condition $T \subseteq S$ is called the *marking condition*. The condition is satisfied when the declaration is sufficiently large.

2.5 Theoretical Problem

The idea of the original extended projection method is similar to program analysis where proof trees are regarded as programs. The marking procedure performs an analysis which is similar to strictness analysis with abstract interpretation. The marking system (marking and the marking procedure) and the marked proof trees are not a part of QPC. Also, no realizability interpretation corresponding to $NExt$ is given. The aim of this paper is to give a logic which embodies the marking system.

3. The Logic: QPC^{KT}

QPC^{KT} is a constructive logic which is an extension of QPC with a new logical constant $\check{\exists}$ describing markings. The marking procedure is described as a transformation of $\exists$ to $\check{\exists}$ in formula occurrences in a proof tree. The extended projection method is internalized in QPC^{KT} as a part of the logic in this respect.

3.1 The Language of QPC^{KT}

Terms in QPC^{KT} are those of a variant of untyped λ-calculus, and they are used to describe the codes extracted from proofs.

Definition 1: Terms

1) [constants] $0, 1, \cdots$ (natural numbers), $left$, $right$, any, nil (nil list), T, F (booleans) are terms;
2) [individual variables] $x, y, z, \cdots$ are terms;
3) [sequence of terms] If $M_1, \cdots, M_n$ are terms , then $(M_1, \cdots, M_n)$ is a term;
() denotes nil sequence. A sequence of variables is often denoted $\overline{x}$. $any[n]$ denotes the sequence $(any, \cdots, any)$ of length n;
4) [λ-term] If M is a term and X is a variable or a sequence of variables, then $\lambda X.M$ is a term;
5) [application] If M and N are terms, then $ap(M, N)$ is a term;
6) [if-then-else] If M and N are terms, and if A is a (in)equality of terms, then $if\ beval(A)\ then\ M\ else\ N$ is a term;
7) [μ-term] If M is a term and Z is a variable or a sequence of variables, then $\mu Z.M$ is a term;
8) [let-sentence] If X is a variable or a sequence of variables, and if T and M are terms, then $let\ X = T\ in\ M$ is a term;

9) [built-in functions] *tseq*, *ttseq*, *proj* and *beval* are terms.

In the following description, it is assumed that the term structure is suitably extended by introducing arithmetic operators and list constructor functions. The meaning of these built-in functions is as follows: Assume that $S \stackrel{\text{def}}{=} (S_1, \cdots, S_n)$ is a sequence of terms of length n, then

$$tseq(k, S) = (S_k, \cdots, S_n) \qquad (1 \leq k \leq n)$$

$$ttseq(k, l, S) = (S_k, \cdots, S_{k+l-1}) \qquad (1 \leq k \leq n, 1 \leq l \leq (n-k+1))$$

$$proj(k, S) = S_k \qquad (1 \leq k \leq n)$$

beval is a function which takes (in)equality as input and returns boolean values, T and F, according to whether the (in)equality holds or not.

Types in QPC^{KT} are only to specify the domains of universally and existentially quantified variables and to categorize terms used in proofs.

Definition 2: Types

1) **2**, *nat* and *bool* (primitive types) are types;
2) If σ and τ are types, then $\sigma \times \tau$ (cartesian product) and $\sigma \to \tau$ (function spaces) are types;
3) If σ is a type, then $L(\sigma)$ (type of lists over σ) is a type.

The formulas in QPC^{KT} are ordinary first order formulas plus special formulas called checked $\exists$-formula.

Definition 3: Formulas

1) If M and N are terms and if σ is a type, then $M = N$, $M : \sigma$ and $\bot$ are (atomic) formulas;
2) If A and B are formulas and σ is a type, then $A \wedge B$, $A \vee B$, $A \supset B$, $\forall x \in \sigma.A$, and $\exists x \in \sigma.A$ are formulas;
3) [checked formula] If A is a formula and σ is a type, then $\dot{\exists} x \in \sigma.A$ is a formula.

The typing relation, $M : \sigma$, is abbreviated to M when σ is clear from the context or is not significant. $\dot{\exists} x \in \sigma.A$ means "I do not need the value of x in the extracted code."

The substitution of a term M to x in an expression, E, is denoted $E_x[M]$, and $E_{x_1,\cdots,x_n}[M_1, \cdots, M_n]$ denotes simultaneous substitution in the following description.

3.2 Rules of Inferences

(1) Rules on first order logic

Introduction and elimination rules on $\vee$, $\wedge$, $\supset$, $\forall$ and $\exists$ are as usual. Equality rules (reflectivity, commutativity, transitivity, and $=$ elimination rule) and $\bot$ elimination rule are also as usual. For the induction rules, there are mathematical induction and structural induction on lists:

$$\frac{A_x[0] \qquad \begin{matrix}[x : nat, A_x[x-1], x > 0] \\ A\end{matrix}}{\forall x \in nat.\ A} \qquad \frac{A_x[nil] \qquad \begin{matrix}[x : L(\sigma), x \neq nil, A_x[tl(x)]] \\ A\end{matrix}}{\forall x \in L(\sigma).\ A}$$

(2) Rules on $\dot{\exists}$

$$\frac{M : \sigma \quad A_x[M]}{\dot{\exists} x \in \sigma.\ A}(\dot{\exists} I) \qquad \frac{\dot{\exists} x \in \sigma . A \qquad \begin{matrix}[x : \sigma, A] \\ C\end{matrix}}{C}(\dot{\exists} E)$$

Besides the ordinary side condition for the elimination rule of existential quantifier, $(\dot{\exists} E)$ has an additional side condition: In the proof of $x : \sigma, A \vdash C$, the eigen variable, x, must not occur in N in the following forms of subproofs:

$$\frac{\begin{matrix}\Sigma_0 & \Sigma_1 \\ N : \sigma & B_y[N]\end{matrix}}{\exists y \in \sigma.\ B}(\exists I) \qquad \frac{\begin{matrix}\Sigma_0 & \Sigma_1 \\ N : \sigma & \forall y \in \sigma.\ B\end{matrix}}{B_y[N]}(\forall E)$$

(3) Rules on Term Calculus

The equality rules for sequences of terms are as follows:

$$ap((M_1, \cdots, M_n), N) = (ap(M_1, N), \cdots, ap(M_n, N))$$

$$\lambda X.(M_1, \cdots, M_n) = (\lambda X.M_1, \cdots, \lambda X.N_n)$$

$$\begin{aligned} &if\ A\ then\ (M_1, \cdots, M_n)\ else\ (N_1, \cdots, N_n) \\ &\quad = (if\ A\ then\ M_1\ else\ N_1, \cdots, if\ A\ then\ M_n\ else\ N_n) \end{aligned}$$

$$let\ X = T\ in\ (M_1, \cdots, M_n) = (let\ X = T\ in\ M_1, \cdots, let\ X = T\ in\ M_n)$$

The reduction rules on terms are defined as usual, and if a term M is reduced to N in finite steps then M and N are regarded as equal. A μ-term, $\mu(z_1, \cdots, z_n).M$ denotes the fixed point of M. If M is equal to a sequence of term, $(M_1, \cdots, M_n)$, then $\mu(z_1, \cdots, z_n).M$ denotes a solution of the following system of equations:

$$\begin{aligned} z_1 &= M_1 \\ &\cdots \\ z_n &= M_n \end{aligned}$$

Let $(F_1, \cdots, F_n)$ be a solution of the equations: $\mu(z_1, \cdots, z_n).(M_1, \cdots, M_n) = (F_1, \cdots, F_n)$. Then, F_i satisfies the following equations:

$$F_i = \mu z_i.(M_i)_{z_1, \cdots, z_{i-1}, z_{i+1}, \cdots, z_n}[F_1, \cdots, F_{i-1}, F_{i+1}, \cdots, F_n] \qquad (1 \leq i \leq n)$$

(See, for example, (Barendregt, 1981) page 138, for multiple fixed point theorem.) Therefore, the followings are the rules for μ-terms:

$$\mu(z_1, \cdots, z_n).(M_1, \cdots, M_n)$$

where F_i $(1 \leq i \leq n)$ is a solution of the fixed point equation.

$$\mu z.M = M_z[\mu z.M]$$

$$\frac{M = (M_1, \cdots, M_n) \quad \mu(z_1, \cdots, z_n).(M_1, \cdots, M_n) = (F_1, \cdots F_n)}{\mu(z_1, \cdots, z_n).M = M_{z_1, \cdots, z_n}[F_1, \cdots, F_n]} \qquad (n \geq 2)$$

For typing rules of terms, the constants, *left* and *right*, are of type **2**. Other rules are defined as usual. Finally, the following rule on type structure should be introduced to assure the unicity of typing:

$$\sigma \to (\tau_1 \times \cdots \times \tau_n) = (\sigma \to \tau_1) \times \cdots \times (\sigma \to \tau_n)$$

For example, if $ap(M, N) : \tau$, then M and N should be of type $\sigma \to \tau$ and σ with some σ. If $M = (M_1, \cdots, M_n)$, then $ap(M, N) = (ap(M_1, N), \cdots, ap(M_n, N))$, so that M_i should be of type $\sigma \to \tau_i$ with some τ_i, so that $M : (\sigma \to \tau_1) \times \cdots \times (\sigma \to \tau_n)$ and $ap(M, N) : (\tau_1 \times \cdots \times \tau_n)$. Therefore, $\sigma \to (\tau_1 \times \cdots \times \tau_n) = (\sigma \to \tau_1) \times \cdots \times (\sigma \to \tau_n)$. The type equality can also be used for the type inference of $\lambda X.(M_1, \cdots, M_n) = (\lambda X.M_1, \cdots, \lambda X.M_n)$.

3.3 QPC and QPC^{KT}

QPC is obtained by removing the checked formulas and the rules on $\tilde{\exists}$ from QPC^{KT}. QPC is the logic in which proofs are described, and QPC^{KT} is the extension for program extraction with optimization in terms of the extended projection method.

4. Realizability

The realizability interpretation given to QPC^{KT} is called **kt**-realizability. The realizability is a variant of **q**-realizability, and is obtained by modifying the realizability given in (Sato, 1985) with the feature of Kreisel-Troelstra realizability.

4.1 **kt**-realizability

A new class of formulas called realizability relations is introduced in QPC^{KT} to define **kt**-realizability.

Definition 4: Realizability relation
A *realizability relation* is an expression in the form of $\overline{a}$ **kt** A, where A is a formula defined in section 3 and $\overline{a}$ is a sequence of variables which does not occur in A. $\overline{a}$ is called a *realizing variables* of A. For a term, M, M **kt** A, which reads "a term, M, realizes a formula, A", denotes $(\overline{a}$ **kt** $A)_{\overline{a}}[M]$, and M is called a *realizer* of A.

In the following, a formula means one defined in section 3. A type is assigned for each formula.

Definition 5: $type(A)$ Let A be a formula. Then, a type of A, $type(A)$, is defined as follows:

1) $type(A) isempty$, if A is atomic;

2) $type(A \wedge B) \stackrel{\text{def}}{=} type(A) \times type(B)$;

3) $type(A \vee B) \stackrel{\text{def}}{=} \mathbf{2} \times type(A) \times type(B)$;

4) $type(A \supset B) \stackrel{\text{def}}{=} type(A) \to type(B)$;

5) $type(\forall x \in \sigma.A) \stackrel{\text{def}}{=} \sigma \to type(A)$;

6) $type(\exists x \in \sigma.A) \stackrel{\text{def}}{=} \sigma \times type(A)$;

7) $type(\dot{\exists} x \in \sigma.A) \stackrel{\text{def}}{=} type(A)$.

Proposition 1: *Let A be a formula with a free variable x. Then, $type(A) = type(A_x[M])$ for any term M of the same type as x.*

Definition 6: **kt**-realizability

1) If A is a Harrop formula, then $()$ **kt** $A \stackrel{\text{def}}{=} A$;

2) $\overline{a}$ **kt** $A \supset B \stackrel{\text{def}}{=} \forall b \in type(A).(A \wedge b$ **kt** $A \supset ap(\overline{a}, b)$ **kt** $B)$

3) $(a, \overline{b})$ **kt** $\exists x \in \sigma.A \stackrel{\text{def}}{=} a : \sigma \wedge A_x[a] \wedge \overline{b}$ **kt** $A_x[a]$

4) $\overline{a}$ **kt** $\forall x \subset \sigma.A \stackrel{\text{def}}{=} \forall x \in \sigma.(ap(\overline{a}, x)$ **kt** $A)$

5) $(z, \overline{a}, \overline{b})$ **kt** $A \vee B \stackrel{\text{def}}{=} (z = left \wedge A \wedge \overline{a}$ **kt** $A) \vee (z = right \wedge B \wedge \overline{b}$ **kt** $B)$

6) $(\overline{a}, \overline{b})$ **kt** $A \wedge B \stackrel{\text{def}}{=} \overline{a}$ **kt** $A \wedge \overline{b}$ **kt** B

7) $\overline{b}$ **kt** $\dot{\exists} x \in \sigma.A \stackrel{\text{def}}{=} \dot{\exists} a \in \sigma.(A_x[a] \wedge \overline{b}$ **kt** $A_x[a])$ where $\overline{b}$ does not contain a free.

For the definition of Harrop formulas, see (Troelstra, 1973).

Proposition 2: *Let A be any formula. Then, if $\overline{a}$* **kt** *A, then $\overline{a} : type(A)$.*

From the definition of realizability, realizing variables can be determined from the construction of the formulas as follows:

Definition 7: Realizing variables: $Rv(A)$

1) $Rv(A) \stackrel{\text{def}}{=} ()$ $\cdots$ if A is Harrop

2) $Rv(A \wedge B) \stackrel{\text{def}}{=} (Rv(A), Rv(B))$

3) $Rv(A \vee B) \stackrel{\text{def}}{=} (z, Rv(A), Rv(B))$ (z is a fresh variable)

4) $Rv(A \supset B) \stackrel{\text{def}}{=} Rv(B)$

5) $Rv(\forall x \in \sigma.\ A) \stackrel{\text{def}}{=} Rv(A)$

6) $Rv(\exists x \in \sigma.\ A) \stackrel{\text{def}}{=} (z, Rv(A))$ (z is a fresh variable)

7) $Rv(\tilde{\exists} x \in \sigma.\ A) \stackrel{\text{def}}{=} Rv(A)$

Definition 8: Length of a formula
The length of a formula A, $l(A)$, is the length of $Rv(A)$ as a sequence of variables.

Proposition 3: *A formula A is Harrop if and only if $l(A) = 0$*

4.2 Soundness of **kt**-realizability

Theorem 1: Soundness of **kt**-realizability
*Assume that A is a formula. If A is proved in QPC^{KT}, then there is a term, T, such that T **kt** A can be proved in QPC^{KT}, $FV(T) \subset FV(A)$ and T is equal to a sequence of terms of length $l(A)$.*

Proof: The proof is performed by induction on the structure of proof trees. Only the construction of T is presented. The other part of the proof is easy.

1) If the proof tree is solely the formula A, then let $T \stackrel{\text{def}}{=} Rv(A)$.

2) If A is a Harrop formula, then let $T \stackrel{\text{def}}{=} ()$.

2) Step case:

Let R be the name of the last rule which is used in the proof tree of A.

Case $R \equiv (\wedge I)$: Assume that $A \equiv B \wedge C$. By the induction hypothesis, there are terms, M and N, such that M **kt** B and N **kt** C. Then, let T be (M, N). T **kt** A can be proved by $(\wedge I)$.

Case $R \equiv (\wedge E)_1$: Assume that $A \wedge B$ is the premise of the R application. Let M be a term such that M **kt** $A \wedge B$, then let $T \stackrel{\text{def}}{=} ttseq(1, l(A))(M)$. T **kt** A can be proved by $(\wedge E)_1$.

Case $R \equiv (\wedge E)_2$: Assume that $B \wedge A$ is the premise. Let M be a term such that M **kt** $B \wedge A$, then let $T \stackrel{\text{def}}{=} tseq(l(B) + 1)(M)$. T **kt** A can be proved by $(\wedge E)_2$.

Case $R \equiv (\vee I)_1$: Assume that $A \equiv B \vee C$ and B be the premise. Let M be a term such that M **kt** B, then let $T \stackrel{\text{def}}{=} (left, M, any[l(C)])$. T **kt** A can be proved by $(\vee I)_1$.

Case $R \equiv (\vee I)_2$: Assume that $A \equiv B \vee C$ and C be the premise. Let M be a term such that M **kt** C, then let $T \stackrel{\text{def}}{=} (right, any[l(B)], M)$. T **kt** A can be proved by $(\vee I)_2$.

Case $R \equiv (\vee E)$: Assume that $B \vee C$ is the first premise of the R application. Let L be a term such that L **kt** $B \vee C$, and M and N be the term which realize A as the second and the third premises of the R application. Note that M and N may contain $Rv(B)$ and $Rv(C)$. Then let $T \stackrel{\text{def}}{=} if\ proj(1)(L) = left\ then\ (let\ Rv(B) = ttseq(2, l(B))(L)\ in\ M)\ else$ $(let\ Rv(C) = tseq(l(B) + 2)(L)\ in\ N)$. T **kt** A can be proved by substituting $ttseq(2, l(B))(L)$ and $tseq(l(B) + 2)(L)$ to $Rv(B)$ and $Rv(C)$ in the proof of M **kt** A and N **kt** A, and by applying the $(\vee E)$ rule and the $(= E)$ rule.

Case $R \equiv (\supset I)$: Assume that $A \equiv B \supset C$ and C be the premise. Let M be a term such that M **kt** C. M may contain $Rv(B)$. Then, let $T \stackrel{\text{def}}{=} \lambda Rv(B).M$. T **kt** A can be proved by $(\forall I)$, $(\supset I)$ and $(= E)$.

Case $R \equiv (\supset E)$: Assume that $B \supset A$ and B are the premises. Let M and N be terms such that M **kt** $B \supset A$ and N **kt** B. Then, let $T \stackrel{\text{def}}{=} ap(M, N)$. T **kt** A can be proved by $(\supset E)$, $(\wedge I)$ and $(\forall E)$.

Case $R \equiv (\forall I)$: Assume that $A \equiv \forall x.B$. Let M be a term such that M **kt** A, then let $T \stackrel{\text{def}}{=} \lambda x.M$. T **kt** A can be proved by $(\forall I)$ and $(= E)$.

Case $R \equiv (\forall E)$: Assume that $N : \sigma$, and $\forall x.B$ are the premises. Let M be a term such that M **kt** $\forall x.B$. Then, let $T \stackrel{\text{def}}{=} ap(M, N)$. T **kt** A can be proved by $(\forall E)$.

Case $R \equiv (\exists I)$: Assume that $A \equiv \exists x \in \sigma.B$. Assume also that $M : \sigma$ and $B_x[M]$ are the premises. Let N be a term such that N **kt** $B_x[M]$. Then, let $T \stackrel{\text{def}}{=} (M, N)$. T **kt** A can be proved by $(\wedge I)$.

Case $R \equiv (\exists E)$: Assume that $\exists x.B$ is the first premise. Let M be a term such that M **kt** $\exists x.B$ and N be a term which realizes A as the second premise. Then, let $T \stackrel{\text{def}}{=} let\ (x, Rv(B)) = M\ in\ N$. T **kt** A can be proved by substituting $proj(1)(M)$ and $tseq(2)(M)$ to x and $Rv(B)$ in the proof of N **kt** A and replacing the occurrences of $proj(1)(M)$ and $tseq(2)(M)$ **kt** B as the hypotheses by the proof of M **kt** $\exists x.B$.

Case $R \equiv (= E)$: Assume that $M = N$ and $A_x[M]$ be the premises. Let L be a term such that L **kt** $A_x[M]$. Then, let $T \stackrel{\text{def}}{=} L$. T **kt** A can be proved by $(= E)$.

Case $R \equiv (\bot E)$: Let $T \stackrel{\text{def}}{=} any[l(A)]$. T **kt** A can be proved by $(\bot E)$.

Case $R \equiv (\tilde{\exists} I)$: Assume that $A \equiv \tilde{\exists} x \in \sigma.\ B$ and that R application is as follows:

$$\frac{\begin{array}{cc}\Sigma_1 & \Sigma_2 \\ M : \sigma & B_x[M]\end{array}}{\tilde{\exists} x \in \sigma.\ B}$$

By the induction hypothesis, the following proof can be constructed:

$$\begin{array}{c}\Sigma_3 \\ N\ \mathbf{kt}\ B_x[M]\end{array}$$

Then, construct the following proof:

$$\dfrac{\Sigma_1 \quad M:\sigma \qquad \dfrac{\Sigma_2 \quad B_x[M] \qquad \Sigma_3 \quad N\ \mathbf{kt}\ B_x[M]}{B_x[M] \wedge N\ \mathbf{kt}\ B_x[M]}}{\check{\exists} a \in \sigma.(B_x[a] \wedge N\ \mathbf{kt}\ B_x[a])}$$

By definition, $\check{\exists} a \in \sigma.(B_x[a] \wedge N\ \mathbf{kt}\ B_x[a]) \equiv N\ \mathbf{kt}\ \check{\exists} x \in \sigma.\ B$. Then, let $T = N$.
Case $R \equiv (\check{\exists} E)$: Assume that R application is as follows:

$$\dfrac{\Sigma_1 \quad \check{\exists} x \in \sigma.\ B \qquad \overset{[x:\sigma,B]}{\Sigma_2 \quad A}}{A}$$

By the induction hypothesis, the following proofs are obtained:

$$\Pi_1 = \begin{array}{c}\Sigma_3 \\ M\ \mathbf{kt}\ \check{\exists} x \in \sigma.\ B\end{array} \qquad\qquad \Pi_2 = \begin{array}{c}\Sigma_4 \\ N\ \mathbf{kt}\ A\end{array}$$

The hypothesis, B, used in Σ_2 is realized by $Rv(B)$. By definition, $M\ \mathbf{kt}\ \check{\exists} x \in \sigma.B = \check{\exists} a \in \sigma.\ (B_x[a] \wedge M\ \mathbf{kt}\ B_x[a])$. Note that x does not occur in N or A by the side condition of the $(\check{\exists} E)$ rule. Therefore, the following proof is obtained:

$$\dfrac{\Sigma_3 \quad M\ \mathbf{kt}\ \check{\exists} x \in \sigma.B \qquad \dfrac{\overset{[a:\sigma,\, B_x[a] \wedge M\ \mathbf{kt}\ B]}{\Sigma \quad (let\ Rv(B) = M\ in\ N) = N_{Rv(B)}[M] \qquad \Sigma_5 \quad N_{Rv(B)}[M]\ \mathbf{kt}\ A}}{(let\ Rv(B) = M\ in\ N)\ \mathbf{kt}\ A}(= E)}{(let\ Rv(B) = M\ in\ N)\ \mathbf{kt}\ A}(\check{\exists} E)$$

Σ_5 is obtained as follows: First replace the occurrences of $Rv(B)\ \mathbf{kt}\ B$ as the hypothesis with

$$\dfrac{B \wedge Rv(B)\ \mathbf{kt}\ B}{Rv(B)\ \mathbf{kt}\ B}(\wedge E)$$

and then substitute M to $Rv(B)$ and a to x in Σ_4. Then, let $T \overset{\text{def}}{=} let\ Rv(B) = M\ in\ N$.
Case $R \equiv (L(\sigma)\ ind)$: Assume that $A \equiv \forall x \in L(\sigma).B$, and A is proved as follows:

$$\dfrac{\Sigma_1 \quad B_x[nil] \qquad \overset{[x \neq nil \wedge B_x[tl(x)]]}{\Sigma_2 \quad B}}{\forall x \in L(\sigma).B}$$

By the induction hypothesis, the following proofs exist:

$$\begin{array}{cc}\Sigma_1' & \Sigma_2' \\ M\ \mathbf{kt}\ B_x[nil] & N\ \mathbf{kt}\ B\end{array}$$

The assumption, $x \neq nil \wedge B_x[tl(x)]$, in Σ_2 is replaced by the realizing relation: $\overline{x}$ **kt** $(x \neq nil \wedge B_x[tl(x)]) = (x \neq nil \wedge \overline{x}$ **kt** $B_x[tl(x)])$ where $\overline{x} \stackrel{\text{def}}{=} Rv(B_x[tl(x)])$. Let $T' \stackrel{\text{def}}{=} if\ x = nil\ then\ M\ else\ N$. Note that N generally contains $\overline{x}$. Then, construct the the following proof:

$$\Pi_b = \frac{\begin{array}{cc}\Sigma_3 & \Sigma_1' \\ T_x'[nil] = M & M \textbf{ kt } B_x[nil]\end{array}}{T_x'[nil] \textbf{ kt } B_x[nil]}(= E)$$

and

$$\Pi_s = \frac{\dfrac{\dfrac{x \neq nil \wedge \overline{x} \textbf{ kt } B_x[tl(x)]}{x \neq nil}(\wedge E)}{T' = N} \qquad \dfrac{\Sigma_2'}{N \textbf{ kt } B}}{T' \textbf{ kt } B}(= E)$$

Let $\overline{z}$ be a new sequence of variables of length $l(B)$, and assume the following equation:

$$\overline{z} = \lambda x.T_{\overline{x}}'[ap(\overline{z}, tl(x))]$$

The solution of this equation is the following recursive call function: $F \stackrel{\text{def}}{=} \mu\overline{z}.\lambda x.if\ x = nil\ then\ M\ else\ N_{\overline{x}}[ap(\overline{z}, tl(x))]$. Note that, by the induction hypothesis, M and N are equal to sequences of terms of length $n(\stackrel{\text{def}}{=} l(B))$. Therefore, F is equal to a sequence of terms $F_1, \cdots, F_n$. Substitute $ap(F, tl(x))$ to $\overline{x}$ in Π_s to obtain Π_s'. $ap(F, x) = T_{\overline{x}}'[ap(F, tl(x))]$ can be proved. Then, an induction proof of $\forall x \in L(\sigma).\ T_{\overline{x}}'[ap(F, tl(x))]$ **kt** B $(= F$ **kt** $\forall x \in L(\sigma).\ B)$ can be constructed with Π_b and Π_s'. Therefore, let $T = F$.

Case $R \equiv (nat\ ind)$: Similar to the previous case.

Let $T \stackrel{\text{def}}{=} \mu\overline{z}.\lambda x.if\ x = 0\ then\ M\ else\ N_{\overline{z}}[ap(\overline{z}, x-1)]$ where $\overline{z}$ is a new sequence of variables of length $l(A)$.

∎

5. Program Extraction Algorithm: EXT

The proof of theorem 1 can be formalized as a program extractor procedure, EXT. It is the same as the Ext procedure given in (Takayama, 1990) for the inference besides $(\tilde{\exists}I)$ and $(\tilde{\exists}E)$.

Definition 9: EXT ($\tilde{\exists}$ part)

1) $(\tilde{\exists}I)$

$$EXT\left(\frac{\begin{array}{cc}\Sigma_1 & \Sigma_2 \\ M : \sigma & A_x[M]\end{array}}{\tilde{\exists}x \in \sigma.\ A}\right) \stackrel{\text{def}}{=} EXT\begin{pmatrix}\Sigma_2 \\ A_x[M]\end{pmatrix}$$

2) $(\tilde{\exists}E)$

$$EXT\left(\frac{\begin{array}{cc} & [x : \sigma, A] \\ \Sigma_1 & \Sigma_2 \\ \tilde{\exists}x \in \sigma.\ A & C\end{array}}{C}\right) \stackrel{\text{def}}{=} let\ Rv(A) = EXT\begin{pmatrix}\Sigma_1 \\ \tilde{\exists}x \in \sigma.A\end{pmatrix}\ in\ EXT\begin{pmatrix}\Sigma_2 \\ C\end{pmatrix}$$

6. Transformation of QPC proofs into QPC^{KT} proofs

Suppose a proof in QPC is given:

$$\frac{\begin{array}{ccc}\Sigma_1 & \cdots & \Sigma_n \\ A_1 & & A_n\end{array}}{B}(Rule_1)$$

First translate B into B' which is made from B by replacing some occurrences of $\exists$ to $\check{\exists}$ in the strictly positive subformulas. The proof transformation, Tr, defined in this section takes the proof of B and B' as inputs and returns a proof of B'. The proof trees in QPC^{KT} generated by Tr corresponds to the marked proof trees in the original extended projection method.

Lemma: *Let A be an arbitrary formula in QPC^{KT}, and A' be a formula which is obtained by replacing some of the occurrences of $\exists$ in strictly positive part by $\check{\exists}$. Then, $A \supset A'$ can be proved in QPC^{KT}.*

Proof: By induction on the construction of A.

Case 1 A is in the form of $\exists x \in \sigma.B$:

Case 1-1: A' is $\check{\exists}x \in \sigma.B'$: Assume that $\exists x \in \sigma.B$ holds. Let $x \in \sigma$ be such that B. By the induction hypothesis, $B \supset B'$ can be proved. Then, $\check{\exists}x \in \sigma.B'$ can be proved by $(\supset E)$, $(\check{\exists}I)$ and $(\exists E)$. Consequently, $\exists x \in \sigma.B \supset \check{\exists}x \in \sigma.B'$ is proved by $(\supset I)$.

Case 1-2: A' is $\check{\exists}x \in \sigma.B'$: Similar to the case 1-1. Use $(\exists I)$ instead of $(\check{\exists}I)$.

Case 2 A is in the form of $\check{\exists}x \in \sigma.B$:

Similar to the case 1-1. Use $(\check{\exists}I)$ and $(\check{\exists}E)$.

Other cases are similar and easy. ∎

The following is the definition of Tr:

$$Tr(A') \stackrel{\text{def}}{=} A' \qquad \text{(assumption)}$$

$$Tr\left(\begin{array}{c}\Sigma \\ A\end{array}\right) \stackrel{\text{def}}{=} \begin{array}{c}\Sigma \\ A\end{array} \qquad \text{where } A \text{ does not contain } \check{\exists}$$

$$Tr\left(\frac{\begin{array}{c}[A] \\ \Sigma \\ B\end{array}}{(A \supset B)'}(\supset I)\right) \stackrel{\text{def}}{=} \frac{\Phi_A\left(Tr\left(\begin{array}{c}\Sigma' \\ B'\end{array}\right)\right)}{A \supset B'}(\supset I)$$

Φ_A is the procedure which replaces every occurrence of A' as hypothesis, which was an occurrence of the hypothesis A in (Σ/B) before Tr, with the proof of $A \vdash A'$ given in the proof indicated in Lemma. Note that because the change of $\exists$ to $\check{\exists}$ is restricted to the strictly positive part of the formula, $(A \supset B)' \equiv A \supset B'$.

$$Tr\left(\begin{array}{c}\Sigma_1 \quad \Sigma_2\\ \dfrac{A \supset B \quad A}{B'}(\supset E)\end{array}\right) \stackrel{\text{def}}{=} \dfrac{Tr\begin{pmatrix}\Sigma_1\\ A \supset B'\end{pmatrix} \quad \Sigma_2}{B'}(\supset E)$$

$$Tr\left(\begin{array}{c}\Sigma_1 \quad \Sigma_2\\ \dfrac{A \quad B}{A' \wedge B'}(\wedge I)\end{array}\right) \stackrel{\text{def}}{=} \dfrac{Tr\begin{pmatrix}\Sigma_1\\ A'\end{pmatrix} \quad Tr\begin{pmatrix}\Sigma_2\\ B'\end{pmatrix}}{A' \wedge B'}(\wedge I)$$

$$Tr\left(\begin{array}{c}[x:\sigma]\\ \Sigma\\ \dfrac{A}{\forall x \in \sigma.\ A'}(\forall I)\end{array}\right) \stackrel{\text{def}}{=} \dfrac{Tr\begin{pmatrix}\Sigma\\ A'\end{pmatrix}}{\forall x \in \sigma.\ A'}(\forall I)$$

$$Tr\left(\begin{array}{c}\Sigma_1 \quad \Sigma_2\\ \dfrac{M:\sigma \quad \forall x.\ A}{A_x[M]'}(\forall E)\end{array}\right) \stackrel{\text{def}}{=} \dfrac{\Sigma_1 \quad Tr\begin{pmatrix}\Sigma_2\\ \forall x.\ A'\end{pmatrix}}{A_x[M]'}(\forall E)$$

where $A_x[M]' \stackrel{\text{def}}{=} A'_x[M]$.

$$Tr\left(\begin{array}{c}\Sigma\\ \dfrac{A}{A' \vee B'}(\vee I)\end{array}\right) \stackrel{\text{def}}{=} \dfrac{Tr\begin{pmatrix}\Sigma\\ A'\end{pmatrix}}{A' \vee B'}(\vee I)$$

$$Tr\left(\begin{array}{c}\Sigma\\ \dfrac{B}{A' \vee B'}(\vee I)\end{array}\right) \stackrel{\text{def}}{=} \dfrac{Tr\begin{pmatrix}\Sigma\\ B'\end{pmatrix}}{A' \vee B'}(\vee I)$$

$$Tr\left(\begin{array}{c}\quad [A] \quad [B]\\ \Sigma_1 \quad \Sigma_2 \quad \Sigma_3\\ \dfrac{A \vee B \quad C \quad C}{C'}(\vee E)\end{array}\right)$$

$$\stackrel{\text{def}}{=} \dfrac{Tr\begin{pmatrix}\Sigma_1\\ \widehat{A} \vee \widehat{B}\end{pmatrix} \quad \Phi_{\widehat{A}}\left(Tr\begin{pmatrix}\Sigma_2\\ C'\end{pmatrix}\right) \quad \Phi_{\widehat{B}}\left(Tr\begin{pmatrix}\Sigma_3\\ C'\end{pmatrix}\right)}{C'}(\vee E)$$

$\widehat{A}$ and $\widehat{B}$ are obtained as follows: Let $A_1, \cdots, A_n$ be the set of occurrences of A' as assumption in $Tr(\Sigma_2/C)$. Note that all the formulas, A_i $(1 \leq i \leq n)$, have the same construct except some occurrences of $\exists$ and $\overset{s}{\exists}$. Then, $\widehat{A}$ is defined as follows:

$$\widehat{A} \stackrel{\text{def}}{=} A_1 \oplus \cdots \oplus A_k$$

$\oplus$ is an associative operator on formulas defined as follows:

1) $A_1 \oplus A_2 \stackrel{\text{def}}{=} A_2$

if A_1 and A_2 are atomic formulas which differ at the most on parameters;

2) $(A_1 \wedge B_1) \oplus (A_2 \wedge B_2) \stackrel{\text{def}}{=} (A_1 \oplus B_1) \wedge (A_2 \oplus B_2)$

3) $(A_1 \vee B_1) \oplus (A_2 \vee B_2) \stackrel{\text{def}}{=} (A_1 \oplus A_2) \vee (B_1 \oplus B_2)$

4) $(A \supset B_1) \oplus (A \supset B_2) \stackrel{\text{def}}{=} (A \supset (B_1 \oplus B_2))$

5) $(\forall x \in \sigma.A_1) \oplus (\forall x \in \sigma.A_2) \stackrel{\text{def}}{=} \forall x \in \sigma.(A_1 \oplus A_2)$

6) $(\exists x \in \sigma.A_1) \oplus (\exists x \in \sigma.A_2) \stackrel{\text{def}}{=} \exists x \in \sigma.(A_1 \oplus A_2)$

7) $(\exists x \in \sigma.A_1) \oplus (\check{\exists} x \in \sigma.A_2) \stackrel{\text{def}}{=} \exists x \in \sigma.(A_1 \oplus A_2)$

$\widehat{B}$ is obtained similarly.

$\Phi_{\widehat{A}}$ and $\Phi_{\widehat{B}}$ are the procedures which replace the occurrences of A' and B' which are made by Tr from the discharged hypothesis, A and B, in Σ_2 and Σ_3 by the proof of $\widehat{A} \vdash A'$ and $\widehat{B} \vdash B'$.

Example 1: Let $A_1 \stackrel{\text{def}}{=} \check{\exists}x.\exists y.\check{\exists}z.\check{\exists}w.B$ and $A_2 \stackrel{\text{def}}{=} \check{\exists}x.\check{\exists}y.\exists z.\check{\exists}w.B$, where B does not contain $\exists$ or $\check{\exists}$. Then, $A_1 \oplus A_2 = \check{\exists}x.\exists y.\exists z.\check{\exists}w.B$.

$$Tr\left(\dfrac{\begin{matrix}\Sigma_1 & \Sigma_2\\ M:\sigma & A_x[M]\end{matrix}}{\exists x \in \sigma.\ A'}(\exists I)\right) \stackrel{\text{def}}{=} \dfrac{\begin{matrix}\Sigma_1 & Tr\left(\begin{matrix}\Sigma_2\\ A_x[M]'\end{matrix}\right)\\ M:\sigma & \end{matrix}}{\exists x \in \sigma.\ A'}(\exists I)$$

$$Tr\left(\dfrac{\begin{matrix}\Sigma_1 & [x:\sigma, A] \\ \exists x \in \sigma.\ A & \Sigma_2 \\ & C\end{matrix}}{C'}(\exists E)\right) \stackrel{\text{def}}{=} \dfrac{Tr\left(\begin{matrix}\Sigma_1\\ Qx \in \sigma.\ \widehat{A}\end{matrix}\right) \quad \Phi_{\widehat{A}}\left(Tr\left(\begin{matrix}\Sigma_2\\ C'\end{matrix}\right)\right)}{C'}(R)$$

where $\widehat{A} \stackrel{\text{def}}{=} \bigoplus_{X \in U} X$ and U is the set of formula occurrences of A' in $Tr(\Sigma_2/C')$ which is made from A in (Σ_2/C) discharged by $(\exists E)$. Q is $\exists$ if there is a following subproof in $Tr(\Sigma_2/C')$, and $\check{\exists}$ otherwise.

$$\dfrac{\begin{matrix}\Sigma_1 & \Sigma_2\\ M:\sigma & B_y[M]\end{matrix}}{\exists y \in \sigma.\ B}(\exists I) \quad (x \in FV(M)) \qquad \dfrac{\begin{matrix}\Sigma_1 & \Sigma_2\\ M:\sigma & \forall y \in \sigma.\ B\end{matrix}}{B_y[M]}(\forall E) \quad (x \in FV(M))$$

Also, $R = (\check{\exists}E)$ if $Q = \check{\exists}$, and $R = (\exists E)$ otherwise. $\Phi_{\widehat{A}}$ is similar to $\Phi_{\widehat{A}}$ and $\Phi_{\widehat{B}}$ in the case of $(\vee E)$.

$$Tr\left(\dfrac{\begin{matrix}\Sigma_1 & \Sigma_2\\ M:\sigma & A_x[M]\end{matrix}}{\check{\exists}x \in \sigma.\ A'}(\exists I)\right) \stackrel{\text{def}}{=} \dfrac{\begin{matrix}\Sigma_1 & Tr\left(\begin{matrix}\Sigma_2\\ A_x[M]'\end{matrix}\right)\\ M:\sigma & \end{matrix}}{\check{\exists}x \in \sigma.\ A'}(\check{\exists}I)$$

$$Tr\left(\begin{array}{c}\Sigma_1 \quad \Sigma_2\\ \dfrac{M=N \quad A_x[M]}{A_x[N]'}(=E)\end{array}\right) \stackrel{\text{def}}{=} \begin{array}{c}\Sigma_1 \quad Tr\left(\begin{array}{c}\Sigma_2\\ A_x[M]'\end{array}\right)\\ \dfrac{M=N}{A_x[N]'}(=E)\end{array}$$

$$Tr\left(\begin{array}{c}\Sigma\\ \dfrac{\bot}{A'}(\bot E)\end{array}\right) \stackrel{\text{def}}{=} \begin{array}{c}\Sigma\\ \dfrac{\bot}{A'}(\bot E)\end{array}$$

$$Tr\left(\begin{array}{c}[A_x[x-1]]\\ \Sigma_1 \quad \Sigma_2\\ \dfrac{A_x[0] \quad A}{\forall x.A'}(nat\ ind)\end{array}\right)$$

$$\stackrel{\text{def}}{=} \begin{array}{c}Tr\left(\begin{array}{c}\Sigma_1\\ A_x[0]'\end{array}\right) \quad \Phi_{\widehat{A_x[x-1]}}\left(Tr\left(\begin{array}{c}\Sigma_2\\ A'\end{array}\right)\right)\\ \hline \forall x.A'\end{array}(nat\ ind)$$

$$Tr\left(\begin{array}{c}[A_x[tl(x)]]\\ \Sigma_1 \quad \Sigma_2\\ \dfrac{A_x[nil] \quad A}{\forall x.A'}(L(\sigma)\ ind)\end{array}\right)$$

$$\stackrel{\text{def}}{=} \begin{array}{c}Tr\left(\begin{array}{c}\Sigma_1\\ A_x[nil]'\end{array}\right) \quad \Phi_{\widehat{A_x[tl(x)]}}\left(Tr\left(\begin{array}{c}\Sigma_2\\ A'\end{array}\right)\right)\\ \hline \forall x.A'\end{array}(L(\sigma)\ ind)$$

Tr may fail as the marking procedure in the original extended projection method may fail. The correspondence between Tr and the marking procedure will be given in the next section.

7. Relation with the Original Extended Projection Method

QPC^{KT} can be regarded as a logical presentation of the original extended projection method: $\tilde{\exists}$ and Tr describe the marking and the marking procedure. The program extraction procedure Ext and $NExt$ are described by EXT. In this section, the correspondence is investigated more formally.

7.1 $\tilde{\exists}$ and marking

In the original version of the extended projection method, programmers can specify which parts of the realizer of a formal specification are necessary by declaring the positions of $\exists$ and $\vee$. Similar interface for programmers can be given to QPC^{KT} system.

Definition 10: Dual Declaration

For a specification, A, a subset of the finite set of natural numbers, $\{1, \cdots, l(A)\}$, is called *(dual) declaration* to A. Each element of a declaration is called a *(dual) marking number*.

The difference from the declaration in the original extended projection method is that the dual declaration specifies which part of the realizer is not necessary in the program extraction.

The *dec* procedure, defined as follows, relates a formula in QPC and its dual declaration to a suitable formula in QPC^{KT} by replacing some occurrences of $\exists$ to $\dot{\exists}$. In other words, the transformation of a formula B to B' explained at the beginning of section 6 can be performed by giving a dual declaration to B.

Definition 11: *dec*

Let A be a formula in QPC and I be a dual declaration to A. Then, a formula, $dec(I, A)$, in QPC^{KT} is defined as follows:

1) $dec(\phi, A) \stackrel{\text{def}}{=} A$;

2) $dec(S, A \wedge B) \stackrel{\text{def}}{=} dec(S(\leq l(A)), A) \wedge dec(S(> l(A)) - l(A), B)$;

3) $dec(S, A \vee B) \stackrel{\text{def}}{=} dec(S(\leq (l(A)+1))-1, A) \vee dec(S(> (l(A)+1))-(l(A)+1), B)$ if $1 \notin S$;

4) $dec(S, A \supset B) = A \supset dec(S, B)$;

5) $dec(\{1\} \cup S, \dot{\exists} x \in \sigma.\ A) \stackrel{\text{def}}{=} \dot{\exists} x \in \sigma.\ dec(S-1, A)$;

6) $dec(S, \exists x \in \sigma.\ A) \stackrel{\text{def}}{=} \exists x \in \sigma.\ dec(S-1, A) \quad if\ 1 \notin S$;

7) $dec(S, \forall x \in \sigma.\ A) = \forall x \in \sigma.\ dec(S, A)$.

where $S(< n) \stackrel{\text{def}}{=} \{x \in S \mid x \leq n\}$, $S(> n) \stackrel{\text{def}}{=} \{x \in S \mid x > n\}$, $S - n \stackrel{\text{def}}{=} \{x - n \mid x \in S\ and\ 1 \leq x - n\}$, and $S + n \stackrel{\text{def}}{=} \{x + n \mid x \in S\}$ for a finite set, S, of positive natural numbers.

Conversely, the *dmn* procedure relates a formula with $\dot{\exists}$ to the dual declaration. Note that for a formula, A, $dmn(A) = \phi$ means that $\dot{\exists}$ does not occur in the strongly positive part of A. Also, let $A \mapsto \widetilde{A}$ be the translation which replaces all the strictly positive occurrence of $\dot{\exists}$ with $\exists$. Then, the set $\{1, 2, \cdots, l(\widetilde{A})\} - dmn(A)$ is the marking of $\widetilde{A}$ in the sense of the original version of the extended projection method.

Definition 12: *dmn*

Let A be a formula in QPC^{KT}, then a subset of $\{1, \cdots, l(\widetilde{A})\}$, $dmn(A)$, is defined as follows:

1) $dmn(A) \stackrel{\text{def}}{=} \phi$ if A is atomic;

2) $dmn(A \wedge B) \stackrel{\text{def}}{=} dmn(A) \cup (dmn(B) + l(\widetilde{A}))$;

3) $dmn(A \vee B) \stackrel{\text{def}}{=} (dmn(A) + 1) \cup (dmn(B) + l(\widetilde{A}) + 1)$;

4) $dmn(A \supset B) \stackrel{\text{def}}{=} dmn(B)$;

5) $dmn(\exists x \in \sigma.A) \stackrel{\text{def}}{=} dmn(A) + 1$;

6) $dmn(\forall x \in \sigma.A) \stackrel{\text{def}}{=} dmn(A)$;

7) $dmn(\check{\exists} x \subset \sigma.A) \stackrel{\text{def}}{=} \{1\} \cup (dmn(A) + 1)$.

Proposition 4: *Let A be a formula in QPC^{KT}. Then, $dec(dmn(A), \widetilde{A}) = A$.*

This means that a formula in QPC^{KT} with $\check{\exists}$ in its strictly positive part can be represented by a formula in QPC and its dual marking. Note that it is possible to specify the realizer corresponding to $\vee$ to be unnecessary in the program extraction in the original extended projection method. However, the redundancy specification here is restricted to $\exists$. Therefore, the *dec* procedure fails if the dual declaration has a marking number which corresponds to $\vee$ in the strictly positive part of the given formula.

7.2 Tr and marking procedure

The following theorem means that the Tr procedure succeeds if and only if $Mark$ succeeds.

Theorem 2: *Let (Σ/A) be a proof tree, and A' be a formula obtained by replacing some of the occurrences of $\exists$ in A with $\check{\exists}$. Also, let (Σ/A') denote the tree obtained by replacing the conclusion A with A' in (Σ/A). Then, $Tr(\Sigma/A')$ succeeds if and only if $Mark(\Sigma/\{A\}_J)$ succeeds where $J \stackrel{\text{def}}{=} \{1, \cdots, l(A)\} - dmn(A')$.*

The marking condition of $Mark$ is reformulated in QPC^{KT} as follows. Assume an induction proof:

$$\dfrac{\begin{array}{cc} & [x \neq 0, A_x[x-1]] \\ \Sigma_1 & \Sigma_2 \\ A_x[0] & A \end{array}}{\forall x \in nat.A}(nat\ ind)$$

Let $B \stackrel{\text{def}}{=} \bigoplus_{X \in U} X$ and U be the set of occurrences of the checked induction hypothesis, $A_x[x-1]'$, in $Tr(\Sigma_2/A')$. The marking condition is described as $B \oplus A' = A'$.

7.3 EXT and $NExt$

The EXT procedure has similar properties to the $NExt$ procedure in terms of projection.

Theorem 3: *Let I be a dual declaration to a proof tree (Σ/A) in QPC and J be $\{1, \cdots, l(A)\} - I$. Then,*

$$EXT\left(Tr\left(\begin{array}{c}\Sigma \\ dec(I,A)\end{array}\right)\right) = proj(J)\left(EXT\left(\begin{array}{c}\Sigma \\ A\end{array}\right)\right)$$

if $Tr(\Sigma/dec(I,A))$ succeeds.

Also, the relation between EXT and $NExt$ is formalized as follows:

Theorem 4: *Let I, J, and (Σ/A) be as in the previous theorem. Then,*

$$EXT\left(Tr\left(\begin{matrix}\Sigma\\ dec(I,A)\end{matrix}\right)\right) = NExt\left(Mark\left(\begin{matrix}\Sigma\\ \{A\}_J\end{matrix}\right)\right)$$

if $Tr(\Sigma/dec(I,A))$ succeeds.

8. Examples

8.1 Even-Odd Checker Program

Assume the following simple theorem which states that every natural number is even or odd.

$$\forall x \in nat.((\exists p \in nat.x = 2 \cdot p) \vee (\exists q \in nat.x = 2 \cdot q + 1))$$

This theorem is proved by mathematical induction on x. The code extracted from the proof is

$$\begin{aligned}\mu(z_1,z_2,z_3).\lambda x.if\ x = 0\ then\ (left, 0, any)\\ else\ if\ proj(1)(ap((z_1,z_2,z_3), x-1)) = left\\ then\ (right, any, ap(z_2, x-1))\\ else\ (left, ap(z_3, x-1)+1, any)\end{aligned}$$

This mutual recursive call function calculates a sequence of terms of length 3. The first element is the constant, $left$ or $right$, which corresponds to $\vee$. The other two elements are values of p and q quantified by $\exists$. This code is a function which checks whether x is even or odd, but the values of p and q are redundant. To generate a redundancy free function, the theorem is translated to the following:

$$\forall x \in nat.((\dot{\exists} p \in nat.x = 2 \cdot p) \vee (\dot{\exists} q \in nat.x = 2 \cdot q + 1))$$

The original proof is translated with Tr as follows:

$$\frac{\Pi_1 \qquad \Pi_2}{\forall x.((\dot{\exists} p.x = 2 \cdot p) \vee (\dot{\exists} q.x = 2 \cdot q + 1))}(ind)^1$$

$$\Pi_1 = \frac{\dfrac{0 : nat \quad 0 = 2 \cdot 0}{\dot{\exists} p.0 = 2 \cdot p}(\dot{\exists} I)}{(\dot{\exists} p.0 = 2 \cdot p) \vee (\dot{\exists} q.0 = 2 \cdot q + 1)}(\vee I)$$

$$\Pi_2 = \frac{[(\dot{\exists} p.(x-1) = 2 \cdot p) \vee (\dot{\exists} q.(x-1) = 2 \cdot q + 1)]^1 \quad \Pi_3 \quad \Pi_4}{(\dot{\exists} p.x = 2 \cdot p) \vee (\dot{\exists} q.x = 2 \cdot q + 1)}(\vee E)^2$$

$$\Pi_3 = \frac{[Hyp_1]^2 \quad \dfrac{\dfrac{[p : nat] \quad \dfrac{[x-1 = 2 \cdot p]}{x = 2 \cdot p + 1}}{\dot{\exists} q.x = 2 \cdot q + 1}(\dot{\exists} I)}{(\dot{\exists} p.x = 2 \cdot p) \vee (\dot{\exists} q.x = 2 \cdot q + 1)}(\vee I)}{(\dot{\exists} p.x = 2 \cdot p) \vee (\dot{\exists} q.x = 2 \cdot q + 1)}(\dot{\exists} E)$$

$Hyp_1 \stackrel{\text{def}}{=} \check{\exists} p.(x-1) = 2 \cdot p$

$$\Pi_3 = \dfrac{[Hyp_2]^2 \quad \dfrac{\dfrac{[q:nat]}{q+1:nat} \quad \dfrac{[x-1=2\cdot q+1]}{x=2\cdot(q+1)}(\check{\exists}I)}{\check{\exists} p.x = 2\cdot p}(\vee I)}{(\check{\exists} p.x = 2\cdot p) \vee (\check{\exists} q.x = 2\cdot q+1)}(\check{\exists}E)$$

$Hyp_2 \stackrel{\text{def}}{=} \check{\exists} q.(x-1) = 2 \cdot q + 1$

The code extracted from this new proof is

$$\mu z.\lambda x.if\ x = 0\ then\ left$$
$$else\ if\ ap(z, x-1) = left\ then\ right\ else\ left$$

On the other hand, if the theorem is changed to the followings, Tr fails.

$$\forall x \in nat.((\exists p \in nat.x = 2\cdot p) \vee (\check{\exists} q \in nat.x = 2\cdot q+1))$$

and

$$\forall x \in nat.((\check{\exists} p \in nat.x = 2\cdot p) \vee (\exists q \in nat.x = 2\cdot q+1))$$

Because the marking condition is not satisfied in both of the cases. It is in fact,

$$\begin{aligned}&((\check{\exists} p \in nat.x-1 = 2\cdot p) \vee (\exists q \in nat.x-1 = 2\cdot q+1))\\ &\oplus((\exists p \in nat.x = 2\cdot p) \vee (\check{\exists} q \in nat.x = 2\cdot q+1))\\ &\neq ((\exists p \in nat.x = 2\cdot p) \vee (\check{\exists} q \in nat.x = 2\cdot q+1))\end{aligned}$$

in the first case, and

$$\begin{aligned}&((\exists p \in nat.x-1 = 2\cdot p) \vee (\check{\exists} q \in nat.x-1 = 2\cdot q+1))\\ &\oplus((\check{\exists} p \in nat.x = 2\cdot p) \vee (\exists q \in nat.x = 2\cdot q+1))\\ &\neq ((\check{\exists} p \in nat.x = 2\cdot p) \vee (\exists q \in nat.x = 2\cdot q+1))\end{aligned}$$

in the second.

8.2 Natural Number Division

A specification for a natural number division program can be written as follows:

$$\forall x \in nat.\forall y \in nat.(y > 0 \supset \exists q \in nat.\exists r \in nat.x = q\cdot y + r \wedge r < y)$$

This specification can be proved by mathematical induction on x. A more elegant proof by course-of-value induction is given in (Hayashi & Nakano, 1988). The extracted program is

$$\begin{array}{l}\mu(z_1,z_2).\lambda x.\ if\ x=0\ then\ \lambda y.(0,0)\\ \qquad else\ \lambda y.\ if\ proj(1)(ap(ap(D,(ap(z_2,x-1)+1)),y))=left\\ \qquad\qquad then\ (ap(z_1,x-1),ap(z_2,x-1)+1)\\ \qquad\qquad else\ (ap(z_1,x-1)+1,0)\end{array}$$

where D is a term extracted from a proof of $\forall a \in nat.\forall b \in nat.(a<b)\vee(b\leq a)$.
This program calculates the pair of the quotient and the remainder of given natural numbers x and y $(y>0)$. If only the remainder is needed, the specification should be translated to

$$\forall x \in nat.\forall y \in nat.(y>0 \supset \dot{\exists} q \in nat.\exists r \in nat.x = q\cdot y + r \wedge r < y)$$

The following program is extracted by Tr and EXT:

$$\begin{array}{l}\mu z_2.\lambda x.\ if\ x=0\ then\ \lambda y.0\\ \qquad else\ \lambda y.\ if\ proj(1)(ap(ap(D,(ap(z_2,x-1)+1)),y))=left\\ \qquad\qquad then\ ap(z_2,x-1)+1\ else\ 0\end{array}$$

However, it is impossible to extract a program which only calculates the quotient. As can be observed from the program extracted from the first specification, the value of remainder at the recursive call step, $ap(z_2,x-1)$, must be used to calculate the quotient. This can also be analyzed at proof tree level: First translate the specification to

$$\forall x \in nat.\forall y \in nat.(y>0 \supset \exists q \in nat.\dot{\exists} r \in nat.x = q\cdot y + r \wedge r < y)$$

Then, the induction step proof obtained from the original proof by Tr procedure is as follows:

$$\dfrac{\dfrac{\dfrac{[y>0]^2 \quad \dfrac{\dfrac{[y]^1 \quad [Hyp]}{y>0\supset\exists q.\exists r.x-1=q\cdot y+r\wedge r<y}(\forall E)}{\exists q.\exists r.x-1=q\cdot y+r\wedge r<y}(\supset E) \quad \Pi_1}{\exists q.\dot{\exists} r.x=q\cdot y+r\wedge r<y}(\exists E)^3}{y>0\supset\exists q.\dot{\exists} r.x=q\cdot y+r\wedge r<y}(\supset I)^2}{\forall y.(y>0\supset\exists q.\dot{\exists} r.x=q\cdot y+r\wedge r<y)}(\forall I)^1$$

where Hyp is the induction hypothesis: $\forall y.(y>0\supset\exists q.\exists r.x-1=q\cdot y+r\wedge r<y)$.

$$\Pi_1 = \dfrac{[\exists r.x-1=q\cdot y+r\wedge r<y]^3 \quad \Pi_2}{\exists q.\dot{\exists} r.x=q\cdot y+r\wedge r<y}(\exists E)^4$$

$$\Pi_2 = \dfrac{\dfrac{[y]^1 \quad \dfrac{\dfrac{[r]^4}{r+1} \quad \forall a.\forall b.a<b\vee b\leq a}{\forall b.(r+1<b\vee b\leq r+1)}(\forall E)}{r+1<y\vee y\leq r+1}(\forall E) \quad \Pi_3 \quad \Pi_4}{\exists q.\dot{\exists} r.x=q\cdot y+r\wedge r<y}(\vee E)$$

where Π_3 and Π_4 are proofs of $\exists q.\check{\exists} r.x = q \cdot y + r \wedge r < y$ under the assumptions, $r + 1 < y$ and $y \leq r + 1$.

Because the eigen variable, r, of the $(\exists E)$ application in Π_1 is used in the $(\forall E)$ application in Π_2, the $(\exists E)$ is not replaced by $(\check{\exists} E)$. Also, the checked $\exists$-formula is not propagated to $\exists r.x - 1 = q \cdot y + r \wedge r < y$ in Π_1. Consequently, the induction hypothesis, Hyp, is not changed by the Tr procedure. Then, the marking condition , $Hyp \oplus A' = A'$ where $A' = \forall y.(y > 0 \supset \exists q.\check{\exists} r.x = q \cdot y + r \wedge r < y)$, is not satisfied, so the Tr procedure fails. This means that the program cannot be extracted.

9. Comparison with Other Works

Application of Kreisel-Troelstra realizability is not new in computer science. J. C. Mitchell and G. Plotkin (1985) used a second order existential quantifier with the realizability to describe module structure in constructive logic. J. L. Krivine and M. Parigot introduced a second order predicate logic called AF_2 (Parigot, 1988) in which universal quantifiers (both first order and second order) are interpreted by the realizability. The main interest in AF_2 is impredicativive second order logic in which various data structure can be defined.
V. Lifschitz (1982) has introduced a formulation of logic similar to QPC^{KT} for another purpose. Roughly, his logic is QPC^{KT} minus universal and existential quantifiers plus checked universal quantifier and a predicate K. First order quantifiers are interpreted as in Kreisel-Troelstra realizability interpretation. Lifschitz defines realizability interpretation of checked universal quantifier and K as

$$r \text{ realizes } K(t) \text{ iff } r = t,$$
$$r \text{ realizes } \check{\forall} x.A \text{ iff } \check{\forall} x.(r \text{ realizes } A).$$

The ordinary quantifiers are defined by checked quantifiers and K as

$$\forall x.A \text{ iff } \check{\forall} x.(K(x) \supset A),$$
$$\exists x.A \text{ iff } \check{\exists} x.(K(x) \wedge A).$$

Note that the realizability interpretation of these defined quantifiers is the same as the ordinary one.
QPC^{KT} must have an extra side condition for $\check{\exists}$ in this paper, but Lifschitz's does not. This is because in Lifschitz's logic existential introduction and universal elimination have the following *logical* side conditions:

$$\frac{A(t) \quad K(t)}{\exists x.A(x)}, \qquad \frac{\forall x.A(x) \quad K(t)}{A(t)}.$$

In Lifschitz's logic, $K(x)$ is not a theorem for any variable x. The counterpart of $K(t)$ in QPC^{KT} will be $\exists x.(x = t)$. $K(t)$ is a theorem for any term t in QPC^{KT}. Then, the

additional side condition on the checked existential elimination rule is inevitable. Therefore, QPC^{KT} is not completely the same as Lifschitz's. It seems that QPC^{KT} is easier to handle than Lifschitz's for our purpose.

QPC^{KT} was introduced by the authors without any knowledge of Lifschitz's work. Later, essentially the same logic as Lifschitz's was suggested to us by J.-Y.Girard, and then G. Mints pointed out that it had been introduced by Lifschitz.

10. Conclusion and Future Work

QPC^{KT} was presented in the traditional formulation of logic. This constructive logic is to give a logical background for the extended projection method developed by one of the authors. Theorems and their proofs are written in QPC, which is a simple first order constructive logic. The proof is, then, translated into QPC^{KT} which is a superset of QPC with a new existential quantifier, $\check{\exists}$. The QPC^{KT} proof obtained by the translation contains additional information about which parts are unnecessary to extract redundancy-free program. The program extractor, EXT, can generate programs both from the original proof and the translated proof, but the program extracted from the translated proof does not have redundancy. This framework is an application of Kreisel-Troelstra realizability. QPC^{KT} handles only the existential quantifier, and does not fully give a logical background for the original extended projection method which also handles redundant code extracted from disjunction formulas. However, if disjunction is defined, following the traditional technique in proof theory, with the existential quantifier as $A \vee B \stackrel{\mathrm{def}}{=} \exists x.(x = 0 \supset A) \wedge (x \neq 0 \supset B)$, then QPC^{KT} covers the full version of the original extended projection method. If disjunction is treated as primitive logical constant, it is necessary to introduce two kinds of disjunction. Assume that a new disjunction symbol, say $\check{\vee}$, is introduced and realizability is defined as a **kt** $A \check{\vee} B \stackrel{\mathrm{def}}{=} (A \wedge a$ **kt** $A) \vee (B \wedge a$ **kt** $B)$. The introduction rule for $\check{\vee}$ can be defined as

$$\frac{A}{A \check{\vee} B} \qquad \frac{B}{A \check{\vee} B}$$

However, there is no decent elimination rule for $\check{\vee}$. A good formulation of an elimination rule for $\check{\vee}$ will be found in a framework in which proofs are objects such as type theories and Sato's SPT (Sato, 1989). Also, the additional side condition for the rules on $\check{\exists}$ can be presented more clearly in such a formulation. Moreover, the forgetful semantics used for $\check{\exists}$ will also be used for other logical constants to obtain new constructive theories. Research in this direction is planned for the future.

Acknowledgment

We thank J.-Y. Girards and G. E. Mints for helpful suggestions and comments. The majority of this work was conducted when one of the authors was at ICOT. We thank to Dr. Hasegawa, chief of the fifth laboratory of ICOT, for the opportunity for this research.

References

Barendregt, H. P. (1981). *The Lambda Calculus, Its Syntax and Semantics*, Studies in logic and the foundation of mathematics, Vol. 103, North-Holland.

Beeson, M. (1985). *Foundation of Constructive Mathematics*, Springer.

Constable, R. L. (1986). *Implementing Mathematics with the Nuprl Proof Development Systems*, Prentice-Hall.

Hayashi, S, Nakano, H. (1988). *PX : A Computational Logic*, The MIT Press.

Hayashi, S. (1986). PX: a system extracting programs from proofs, *Proceedings of 3rd Working Conference on the Formal Description of Programming Concepts*, North-Halland.

Howard, W.A. (1980). The Formulas-as-types Notion of Construction, *Essays on Combinatory Logic, Lambda Calculus and Formalism*, eds. J. P. Seldin and J. R. Hindley, Academic Press.

Kreisel, G., Troelstra, A. S. (1979). Formal systems for some branches of intuitionistic analysis, Annals of Mathematical Logic 1.

Lifschitz, V. (1982). Constructive Assertions in An Extension of Classical Mathematics, The Journal of Symbolic Logic, Vol. 47, No. 2.

Mitchell, J. C., Plotkin, G. (1985). Abstract data types have existential type, *Proceedings of 12th Annual Symposium on Principles of Programming Languages*, ACM.

Nordström, B., Petersson, K. (1981). Programming in constructive set theory: some examples, *Proceedings of 1981 Conference on Functional Programming Language and Computer Architecture*, ACM.

Parigot, M. (1988). Programming with Proofs: A Second-Order Type Theory, *Proceedings of European Symposium on Programming 88*, LNCS 300, Springer.

Paulin-Mohring, C. (1989). Extracting F_ω's Programs from Proofs in the Calculus of Constructions, *Proceedings of 16th Annual ACM Symposium on Principles of Programming Languages*, ACM.

Sato, M. (1985). *Typed Logical Calculus*, Technical Report 85-13, Department of Information Science, University of Tokyo.

Sato, M. (1989). Symbolic Proof Theory, manuscript

Takayama, Y. (1988). QPC: **QJ**-based proof compiler – Simple Examples and Analysis, *Proceedings of European Symposium on Programming '88*, LNCS 300, Springer.

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.

Takayama, Y. (1990). Extraction of Redundancy-free Programs from Constructive Natural Deduction Proofs, Journal of Symbolic Computation (to appear)

Troelstra, A. S. (1973). *Mathematical investigation of intuitionistic arithmetic and analysis*, Lecture Notes in Mathematics 344, Springer.