TR－420

構成的証明から冗長性の無いプログラム導出する証明論的手法

高山 幸秀

(財) 新世代コンピュータ技術開発機構
〒108 東京都港区三田 1-4-28 三田国際ビルヂング 21 階
takayama@icot.jp

実現可能解釈 (realisability interpretation) によって構成的証明からプログラムが導出されることは古くから知られている。しかしながら、実現可能解釈では実用上意味のない処理を行なう冗長なコードを含んだプログラムが導出されることが多い。本論文では、証明木を解析し、各ノードに付加情報を付けることにより、冗長性の無いプログラムを導出する手法 (Extended projection method) を与える。
基本的な考え方は、まず証明図の結論部分にある種の宣言を行って∃で束縛された変数のうちどの変数の値が必要かという情報を与える。次に、通常の実現可能解釈の場合にたどる証明図の経路に沿ってその情報を伝搬させてゆき、各ノードに同様の情報を与える。この操作は、結論部分の情報とそこで使われる推論規則によって決定的に行なえる。これをマーキングと呼ぶ。実現可能解釈はこの情報を参照して不要なコードを生成しないようにする。この方法は、帰納法を使った証明の場合にはかなり複雑になる。本論文では、このような場合の解析方法も与える。

Proof Theoretic Approach to the Extraction of Redundancy-free Realiser Codes

Yukihide Takayama

Institute for New Generation Computer Technology
4-28, Mita 1-chome, Minato-ku, Tokyo 108 Japan
takayama@icot.jp

Executable codes can be extracted from constructive proofs by using realisability interpretation. However, realisability also generates the redundant codes that have no significant computational meaning. The redundancy causes heavy runtime overhead, and is one of the obstacles in applying realisability to practical systems that realize the mathematical programming paradigm. This paper presents a proof theoretic method to eliminate redundancy by analyzing proof trees as pre-processing of realisability interpretation; according to the declaration given to the theorem that is proved, each node of the proof tree is marked automatically to show which part of the realiser is needed. This procedure does not always work well. This paper also gives an analysis of it and technique to resolve critical cases. The method is studied in a simple constructive logic with primitive types, mathematical induction and its q-realisability interpretation.

1. Introduction

The idea of program synthesis from constructive proofs by using the notion of realisability is rather old. However, the programs extracted by realisability, *realiser codes*, are not always efficient: they generally contain some redundancy. The classification of the redundancy is given in [Takayama 88a].
[Bates 79] applied a traditional syntactical optimization technique on the code extracted from proofs which might destroy the clear correspondence between proofs and program via realisability. [Sasaki 86] improved the program extraction algorithm based on realisability so that the trivial code for the formulae that has no significant computational meaning can be simplified. A similar technique is used in the PX system [Hayashi 87] as *type 0 formulas*. The QPC system [Takayama 88a] uses a similar technique to Sasaki's, normalization method to eliminate β-redex in the extracted codes, and *modified* $\vee$ *code* technique to simplify certain classes of decision procedures. However, the code extracted from constructive proofs still has redundancy. That is the codes which can be seen as verification information of the extracted algorithms.
The most reasonable idea to overcome this problem would be introducing suitable notation to specify which part of the proof is necessary in terms of computation. The set notation $\{x : A|B\}$ is introduced in the Nuprl system [Constable 86] as a weaker notion of $\exists x : A.\ B$. This is to skip the extraction of the justification for B. [Mohring 88] modified *the calculus of construction* [Coquand 86] by introducing two kinds of constants, *Prop* and *Spec*, to distinguish the formulae in proofs whose computational meaning is not necessary. These works are performed in the Martin-Löf's style of constructive type theory.
This paper presents a proof theoretic method in the style of D. Prawitz to perform the program analysis at proof tree level, and to generate redundancy-free realiser code. In some cases, the redundancy can be removed easily by applying a projection function to the extracted code. However, the situation around the redundancy is a little more complicated particularly when the program extraction is performed on the proofs which use induction. The method of program analysis can be presented quite clearly and naturally if it is performed at the proof tree level because proofs are the logical description of programs and have a lot of information about the programs.
The notion of marking, which corresponds to the set notation in the Nuprl and *Spec* and *Prop* constants in the calculus of construction, is introduced, and the program extraction algorithm is also given.

2. Proof theoretic terminology and notation

Following is the list of the notation and terminologies used in this paper. Most of them are borrowed from [Prawitz 65].

Definition 1: *Basic notation*
(1) Π denotes a proof tree, and Σ denotes a sequence of proof trees;
(2) Proof trees such as follows are denoted $([A]/\Sigma/B)$.

$$\begin{array}{c}[A]\\ \Sigma \\ \hline B\end{array}$$

Definition 2: *Proof theoretic terminologies*
(1) *Application*, see [Prawitz 65] p26;
(2) Formulae which occur as conclusions and premises of applications of rules are called *nodes*;
(3) *The subtree of a tree, Π, determined by a formula, A*, see [Prawitz 65] p25;
(4) *Top and end-formula*, see [Prawitz 65] p25;
(5) *Side-connected*, see [Prawitz 65] p26;
(6) *Minor and major-premise*, see [Prawitz 65] p22;
(7) An application of ($\supset$-I) succeeded by an application of ($\supset$-E) is called a *cut*.
(8) *Thread*, see [Prawitz 65] p25;
(9) *Segment*, see [Prawitz 65] p49;
(10) *Path*, see [Prawitz 65] p52;
(11) *Main path*, see [Prawitz 65] p53;

3. A Program Extractor System: QPC

QPC is a sugared version of a subset of QJ [Sato 86]. It is the subset of QJ which is related to program extraction from proofs, and the primitive data structure is restricted to natural numbers and lists of natural numbers. QPC is, roughly, an intuitionistic version of natural deduction with mathematical induction and induction on natural number lists plus higher order equality and inequality. The program part of QPC is given as ordinary lambda calculus with sequences of terms and multi-valued recursive call programs. See [Takayama 88a] for more detail.

3.1 q-realisability and *Ext* procedure

Definition 3: q-*realisability*
Let $x, y, \cdots$ denote variables. and $\overline{x}, \overline{y}, \cdots$ denote sequences of variables. $\overline{x}$ means $x_1 \wedge \cdots \wedge x_m$ where. x_i is an abbreviation of $x_i : \sigma$ for a type σ.

1. If A is an atomic formula, then a q $A \overset{\text{def}}{=} A \wedge a = nil$
2. $(\overline{x}, \overline{y})$ q $A \wedge B \overset{\text{def}}{=} \overline{x}$ q $A \wedge \overline{y}$ q B
3. $(z. \overline{x}, \overline{y})$ q $A \vee B \overset{\text{def}}{=} (\mathbf{outl}(z) \wedge A \wedge \overline{x}$ q $A \wedge \overline{y} \downarrow) \vee (\mathbf{outr}(z) \wedge \overline{x} \downarrow \wedge B \wedge \overline{y}$ q $B)$
4. $\overline{y}$ q $A \supset B \overset{\text{def}}{=} \overline{y} \downarrow \wedge \mathbf{mono}(\overline{y}) \wedge \forall \overline{x}.(A \wedge \overline{x}$ q $A \supset \overline{y}(\overline{x})$ q $B)$
5. $\overline{y}$ q $\forall x.A \overset{\text{def}}{=} \forall x.\ (\overline{y}(x)$ q $A)$
6. $(z, \overline{y})$ q $\exists x.A \overset{\text{def}}{=} z \wedge A_x[z] \wedge \overline{y}$ q $A_x[z]$

where $\mathbf{mono}(\overline{y}) \overset{\text{def}}{=} \forall \overline{x}_0.\forall \overline{x}_1.(\overline{x}_0 \sqsubseteq \overline{x}_1 \supset \overline{y}(\overline{x}_0) \sqsubseteq \overline{y}(\overline{x}_1))$. $A_x[z]$ means the substitution of z to x which occurs freely in A.

In the standard q-realisability, a q $A \overset{\text{def}}{=} A$ if A is atomic. However, the realisers for atomic formulae are restricted to *nil* in QJ. The realisability here is not the standard q-realisability in this respect.

Ext procedure is the algorithmic version of the realisability interpretation given in [Takayama 88a] which takes constructive proofs as inputs and returns the computational meaning of the proofs in λ-expressions.

3.2 Realizing variables and length of formulae

The *realizing variable sequence* (or simply *realizing variables*) for a formula, A, which is denoted $Rv(A)$, is a sequence of variables to which realiser codes for the formula are assigned. Realizing variable sequences are used as the realiser code for assumption in the reasoning of natural deduction.

Definition 4: $Rv(A)$

1. $Rv(A) \overset{\text{def}}{=} (nil)$, if A is atomic.
2. $Rv(A \wedge B) \overset{\text{def}}{=} (Rv(A), Rv(B))$.
3. $Rv(A \vee B) \overset{\text{def}}{=} (z, Rv(A), Rv(B))$ where z is a new variable.
4. $Rv(A \supset B) \overset{\text{def}}{=} Rv(B)$.
5. $Rv(\forall x : \sigma.\ A(x)) \overset{\text{def}}{=} Rv(A(x))$.
6. $Rv(\exists x : \sigma.\ A(x)) \overset{\text{def}}{=} (z, Rv(A(x)))$ where z is a new variable.

Definition 5: *Length of formulae*
$l(A)$, which is called the *length of formula* A, is the length of $Rv(A)$.

4. Declaration to specifications

The declaration indicates which values of the existentially quantified variables of a given theorem are needed.

Definition 6: *Declaration*
(1) A *declaration* of a specification, A, is the finite set, I, of offsets of $Rv(A)$. It is a subset of the set of natural numbers totally ordered by $\leq$. A specification, A, with the declaration, I, is denoted $\{A\}_I$. Elements of the declaration are called *marking numbers*.
(2) The empty set, ϕ, is called the *nil declaration*.
(3) The declaration, $\{0, 1, \cdots, l(A) - 1\}$, is called *trivial*.

Suppose, for simplicity, that the given theorem is of the following canonical form:

$$\forall x_0. \cdots \forall x_{m-1}.\exists y_0. \cdots \exists y_{n-1}.A(x_0, \cdots, x_{m-1}, y_0, \cdots, y_{n-1}),$$

and the values of $y_0, \cdots, y_k$, $0 \leq k \leq n-1$, are needed. It is declared with the set of the positions:

$$\{0, \cdots, k\}$$

The following restriction assures a sort of soundness.

Restriction: The marking numbers of a declarations cannot specify realizing variables of more than two subformulae of the specification which are separated by $\wedge$.

5. Marking

Marking means to attach to each node of given proof trees the information that indicates which codes among the realiser sequence of a given formula are needed. The marking can be determined according to the inference rule of each node and the declaration as will explained later.

Definition 7: *Marking*
(1) *Marking* of a node, A, in a proof tree, Π, is the finite set, I, of offsets of $Rv(A)$. It is a subset of the set of natural numbers totally ordered by $\leq$. A node, A, with the marking, I, is denoted $\{A\}_I$. Elements of the marking are called *marking numbers*.
(2) The empty set, ϕ, is called *nil marking*.
(3) The marking, $\{0, 1, \cdots, l(A)-1\}$, is called *trivial marking*.

Note that declaration is a special case of marking; the marking of the end-formula of the proof tree is the declaration.

Definition 8: *Marked proof tree*
A *marked proof tree* is a tree obtained from a proof tree and the declaration by the marking procedure.

The marking procedure continues from the bottom of proof trees to the tops. The proof compilation procedure, Ext, should be modified to take marked proof trees as inputs and extract part of the realiser code according to the marking. It will be defined later. The formal definition of the marking procedure, called $Mark$, is given in [Takayama 88b], but here, part of the definition will be given rather informally to make the idea clearer.

5.1 Marking of the ($\exists$-I) application

By definition, the 0th code of

$$Ext\left(\dfrac{\overline{t}^{(*)} \quad \dfrac{\Sigma}{A(t)}}{\exists x.A(x)}(\exists\text{-}I)\right)$$

is the term which is the value of x bound by $\exists$. Let I be the marking of the conclusion, then t should be marked $\{0\}$ if $0 \in I$, otherwise the marking is ϕ. The marking of $A(t)$ is given as all marking numbers in I except 0. However, note that the ith code $(0 < i)$ of $\exists x.A(x)$ corresponds to the $i-1$th code of $A(t)$. Consequently, the marking of $A(t)$ is $(I - \{0\}) - 1$ where, for any finite set of natural numbers, K, and any natural number, n, $K - n \overset{\mathrm{def}}{=} \{a - n | a \in K \wedge n \leq a\}$. $K + n$ is defined similarly.

5.2 Marking of the ($\exists$-E) application

By the definition of the Ext procedure, the realiser code of C concluded by the following inference is obtained by instantiating $Rv(A(t))$ in the code from the subtree, $([t, A(t)]/\Sigma_1/C)$, by the code from the subtree, $(\Sigma_0/\exists x.A(x))$:

$$\dfrac{\dfrac{\Sigma_0}{\exists x.\ A(x)} \qquad \dfrac{\begin{matrix}[t, A(t)]\\ \Sigma_1\end{matrix}}{C}}{C}(\exists\text{-}E)$$

where $A(x)$ contains x as free variables.
Hence, both the marking of C as the conclusion of the above tree and the marking of C as the minor premise are the same. The marking of the subtree determined by the minor premise can be performed inductively. Let J and K be the union of the marking of all occurrences of the two hypotheses, t and $A(t)$. Note that J is either $\{0\}$ or ϕ.

$$\dfrac{\dfrac{\Sigma_0}{\exists x.\ A(x)} \qquad \dfrac{\begin{matrix}[\{t\}_J, \{A(t)\}_K]\\ \Sigma_1\end{matrix}}{\{C\}_I}}{\{C\}_I}(\exists\text{-}E)$$

The marking of the subtree determined by the major premise is as follows:
Case 1: $J = \{0\}$
This means that the following reasoning is contained in the subtree determined by the minor premise:

$$\dfrac{(t) \quad P(t)}{\exists y.\ P(y)}(\exists\text{-}I)$$

and the marking of (t) is $\{0\}$, so that t should be extracted from the proof tree determined by the minor premise, C. Consequently, the 0th element of the sequence of realiser codes of $\exists x.\ A(x)$, which is the value of x in $A(x)$, is necessary to instantiate the code from the subtree determined by the minor premise, so that the marking is:

$$\frac{\Sigma_0}{\{\exists x.\ A(x)\}_{\{0\}\cup(K+1)}}$$

Case 2: $J = \phi$
This means that the value of x is not necessary to instantiate the code from the subtree determined by the minor premise, so that the marking is:

$$\frac{\Sigma_0}{\{\exists x.\ A(x)\}_{K+1}}$$

5.3 Marking of the ($\vee$-E) application

The realiser code of C concluded by the following inference

$$\dfrac{\dfrac{\Sigma_0}{A \vee B} \quad \dfrac{\begin{matrix}[A]\\ \Sigma_1\end{matrix}}{C} \quad \dfrac{\begin{matrix}[B]\\ \Sigma_2\end{matrix}}{C}}{C}\,(\vee\text{-}E)$$

is an *if* T_0 *then* T_1 *else* T_2 type code where T_1 and T_2 are sequences of the same length (because both are the codes of C), so that C as the conclusion and two Cs as minor premises should have the same marking. T_1 and T_2 are obtained by instantiating $Rv(A)$ and $Rv(B)$ in the code extracted from the subtrees determined by the minor premise. The code extracted from the subtree determined by the major premise is used both to make T_0 and for the instantiation of $Rv(A)$ and $Rv(B)$. Let I be the marking of the conclusion, then the marking of the subtrees determined by the minor premises can be determined inductively. Let J_0 and J_1 be the unions of markings of all As and Bs as hypotheses:

$$\dfrac{\dfrac{\Sigma_0}{A \vee B} \quad \dfrac{\begin{matrix}\{[A]\}_{J_0}\\ \Sigma_1\end{matrix}}{\{C\}_I} \quad \dfrac{\begin{matrix}\{[B]\}_{J_1}\\ \Sigma_2\end{matrix}}{\{C\}_I}}{\{C\}_I}\,(\vee\text{-}E)$$

The marking of the subtree determined by $A \vee B$ is as follows:
Case 1: $I = \phi$
This means that it is not necessary to extract any code from this proof tree, so that, of course, no code from the subtree is necessary:

$$\frac{\Sigma_0}{\{A \vee B\}_\phi}$$

Case 2: $I \neq \phi$
Code T_0 is the decision procedure that decides which formula in A and B actually holds. This is obtained in the 0th code of the sequence of realiser codes of the subtree determined by $A \vee B$. Also, the codes to be assigned to $\{[A]\}_{J_0}$ and $\{[B]\}_{J_1}$ are obtained in the remainder of the code from the subtree, so that the marking is:

$$\frac{\Sigma_0}{\{A \vee B\}_{\{0\}\cup J_0'\cup J_1'}}$$

where $J_0' = J_0 + 1$ and $J_1' = J_1 + l(A)$.

5.4 Marking of the ($\supset$-E) rule

The realiser code of $A \supset B$ is of the following form:

$$\lambda\overline{x}.\ (t_0, \cdots, t_k) \equiv (\lambda\overline{x}.t_0, \cdots, \lambda\overline{x}.t_k)$$

and $(t_0, \cdots, t_k)$ is the code of $A \supset B$ which contains the variable sequence $\overline{x}(= Rv(A))$ as free variables, so that the length of the code from $A \supset B$ is the same as that of B. Let I be the marking of the conclusion. Then, the marking of $A \supset B$ should also be I:

$$\dfrac{\dfrac{\Sigma_0}{A} \quad \dfrac{\Sigma_1}{\{A \supset B\}_I}}{\{B\}_I}\,(\supset\text{-}E)$$

The marking of the subtree determined by A is as follows.
Case 1: The application of $(\supset\text{-}E)$ is part of the cut:
The realiser code of A as the minor premise is restricted by the marking of A as a hypothesis used in the subtree determined by $A \supset B$. Let I be the marking of B, and let J be the union of the marking of As as a hypothesis:

$$\dfrac{\begin{array}{c}\Pi_0\\ A\end{array} \quad \dfrac{\begin{array}{c}\{[A]\}_J\\ \underline{\Sigma_1}\\ B\end{array}}{\{A \supset B\}_I}(\supset\text{-}I)}{\{B\}_I}(\supset\text{-}E)$$

Hence, the marking of the subtree is:

$$\dfrac{\Sigma_0}{\{A\}_J}$$

Case 2: Cut-free proof
The marking of $A \supset B$ restricts only the length of output sequence $\lambda\overline{x}.\ (t_0, \cdots, t_k)$, and, for the input, all the values of the variable sequence $\overline{x}$ are necessary. Specifically, it may happen that some variables in $\overline{x}$ are not used in a particular output subsequence, $\lambda\overline{x}.(t_{i_0}, \cdots, t_{i_l})$, $\{t_{i_0}, \cdots, t_{i_l}\} \subset \{t_0, \cdots, t_k\}$. These redundant variables cannot be detected by the proof theoretic method. However, this cannot always be seen as redundancy; $\lambda(x,y).x$ and $\lambda x.x$ is to be seen as a different function. Consequently, the marking of the subtree determined by the minor premise is trivial.

6. Critical applications

6.1 Induction hypothesis and marking

The programs extracted from induction proofs are recursive call programs. For simplicity, it is assumed in the following description that induction steps are proved without any application of another induction, and induction always means mathematical induction here. If the recursive call program, f, extracted from the induction proof

$$\dfrac{\dfrac{\Sigma_0}{A(0)} \quad \dfrac{\begin{array}{c}[A(x)]\\ \Sigma_1\end{array}}{A(x+1)}}{\forall x.A(x)}(nat\text{-}ind)$$

is a program that calculates a sequence of terms of length $n(= l(\forall x.A(x)))$, every recursive call of f must calculate the sequence of realiser codes of the same positions, so that the marking of not only $A(0)$, $A(x+1)$ (conclusion of the induction step) and $\forall x.A(x)$ but also $A(x)$ (induction hypothesis) should be the same. This raises a question: are the markings of $A(x+1)$ (conclusion of the induction step) and $A(x)$ (hypothesis of induction) by the *Mark* procedure always the same? Actually, if the $(\vee\text{-}E)$, $(\exists\text{-}E)$, $(\supset\text{-}E)$ and $(\wedge\text{-}I\&E)$ rules are used in the proof of induction step, the answer is not always affirmative.
The rest of this section is dedicated to the analysis of these critical applications of the rules.

6.2 Critical $(\vee\text{-}E)$ and $(\exists\text{-}E)$ applications

Let $A(x) \overset{\text{def}}{=} \exists y.\ B(x,y) \vee C(x,y)$ where $B(x,y)$ and $C(x,y)$ are some formulae with x and y as free variables. Suppose that $\forall x.\ A(x)$ is proved by mathematical induction, and the induction step proceeds as follows. $\exists y.\ B(x,y) \vee C(x,y)$ is the induction hypothesis.

$$\dfrac{[\exists y.\ B(x,y) \vee C(x,y)] \quad \dfrac{[B(x,t) \vee C(x,t)] \quad \dfrac{\begin{array}{c}[t]\\ [B(x,t)]\\ \Sigma_0\end{array}}{A(x+1)} \quad \dfrac{\begin{array}{c}[t]\\ [C(x,t)]\\ \Sigma_1\end{array}}{A(x+1)}}{A(x+1)}(\vee\text{-}E)}{A(x+1)}(\exists\text{-}E)$$

If the declaration of $\forall x.\ A(x)$ is $\{0\}$, the marked proof tree is as follows:

$$\dfrac{\{[\exists y.\ B(x,y) \vee C(x,y)]\}_L \quad \dfrac{\{[B(x,t) \vee C(x,t)]\}_K \quad \dfrac{\begin{array}{c}\{[t]\}_P\{[B(x,t)]\}_I\\ \Sigma_{00}\end{array}}{\{A(x+1)\}_{\{0\}}} \quad \dfrac{\begin{array}{c}\{t\}_Q\{[C(x,t)]\}_J\\ \Sigma_{11}\end{array}}{\{A(x+1)\}_{\{0\}}}}{\{A(x+1)\}_{\{0\}}}(\vee\text{-}E)}{\{A(x+1)\}_{\{0\}}}(\exists\text{-}E)$$

where Σ_{00} and Σ_{11} are the suitably marked versions of Σ_0 and Σ_1. I and J are the union of the markings of $B(x,t)$ and $C(x,t)$, and P and Q are the union of the markings of t as hypotheses. Note that P and Q are either $\{0\}$ or ϕ. Then K and L are as follows:
Case 1: $P \cup Q = \{0\}$

$$\begin{aligned} K &= \{0\} \cup (I + 1) \cup (J + l(B(x,t)) \\ L &= \{0\} \cup (K + 1) = \{0,1\} \cup (I + 2) \cup (J + l(B(x,t)) + 1) \end{aligned}$$

Case 2: $P \cup Q = \phi$

$$\begin{aligned} K &= \{0\} \cup (I + 1) \cup (J + l(B(x,t)) \\ L &= K + 1 = \{1\} \cup (I + 2) \cup (J + l(B(x,t)) + 1) \end{aligned}$$

On the other hand, because $\exists y.\ B(x,y) \vee C(x,y)$ is the induction hypothesis, it should have the same marking as $\forall x.\ A(x)$, i.e., $\{0\}$. However, the marking of the induction hypothesis, L, contains a 1 that is not contained in the marking of $\forall x.\ A(x)$. This indicates the fact that it is necessary to specify more codes in the realiser sequences than one expects when ($\vee$-E) and ($\exists$-E) is used below the deduction sequence down from the induction hypotheses.
The reason for this phenomenon is that the realiser code of $A \vee B$ consists not only of the code of A and B but also of the *left* or *right* code, so that the marking of $A \vee B$ must contain 0 except in a few special situations. A similar thing can be said about the marking of $\exists x.A(x)$ type formulae.

Note that all formula occurrences in a segment are of the same form. Any formula occurrence A in a proof tree Π that is not a conclusion or a minor premise of the application of ($\vee$-E) or ($\exists$-E) is a segment. This kind of segment will be called a *trivial segment* in the following description.

Definition 9: *Major premise attached to a formula*
The major premise of the application of ($\vee$-E) or ($\exists$-E) that is side-connected with a formula A in a segment is, if it exists, called the *major premise attached to* A.

Definition 10: *Proper segment*
The segment in a marked proof tree Π is called *proper* iff every formula occurrence in the segment has non-trivial marking.

Definition 11: *Critical segment*
Let Π be a subtree of the induction step proof in a proof tree in induction. A proper segment, σ, in Π is *critical* iff there is a formula occurrence, A, in σ such that the major premise, B, attached to A is a formula occurrence in one of the main paths of Π from the induction hypothesis.

6.3 Critical ($\supset$-E) applications

Suppose that the induction hypothesis is used as a hypothesis above a minor premise of ($\supset$-E) and the proof is cut-free:

$$\begin{array}{c} \dfrac{\dfrac{\begin{array}{c}[A(x)] \\ \Sigma_0\end{array}}{B} \qquad \dfrac{\Sigma_1}{B \supset C}}{C}\,(\supset\text{-}E) \\ \Pi \\ A(x+1) \end{array}$$

Then the marking of B is trivial, so that $[A(x)]$ has trivial marking. In this case, the correspondence between the markings of induction hypotheses and the conclusions of the induction step holds only if the marking of $A(x+1)$ is trivial.

Definition 12: *Critical* ($\supset$-E) *application*
If there is a path from the induction hypothesis to a minor premise, A, of an application of ($\supset$-E), A is called the *critical* ($\supset$-E) *premise*, and the application is called the *critical* ($\supset$-E) *application*.

6.4 Critical ($\wedge$-$I\&E$) applications

Assume that the induction hypothesis is of the form $A \wedge B$ and the end-formula of the proof is $A' \wedge B'$. A and A' are of the same construction and differ at most in some atomic formulae. B and B' are of the same relation. Assume that the proof is as follows:

$$\dfrac{\begin{array}{c} \dfrac{[A \wedge B]}{A} \\ \Pi_0 \\ A' \end{array} \qquad \begin{array}{c} \Sigma_1 \\ B' \end{array}}{A' \wedge B'}$$

Let I be the non-nil marking of $A' \wedge B'$, and assume that $\{a | a \in I \wedge l(A') \leq a\} = I$. Then the marking of A' is ϕ so that the marking of the induction hypothesis, $A \wedge B$, is also ϕ, i.e., different from I. This situation is problematic in terms of the correspondence of markings of induction hypotheses and conclusions of the induction steps. The restriction on declarations in Section 4 prevents this sort of situation.

6.5 Soundness of the marking procedure

Theorem 1: [Takayama 88b]
Suppose that a formula, $\forall x.A(x)$, is proved by mathematical induction, and that I is an arbitrary declaration of the conclusion. Let Π be a normal deduction of $A(x) \vdash A(x+1)$, and assume that there is no critical ($\wedge$-I&E) application in Π:

$$\dfrac{A(0) \qquad \dfrac{\begin{matrix}[A(x)]\\ \Sigma\end{matrix}}{A(x+1)}}{\forall x.\ A(x)}(nat\text{-}ind)$$

(1) If Π has a critical ($\supset$-E) application in one of the main paths from the induction hypothesis, $[A(x)]$, its marking is trivial.
(2) If Π has no critical ($\supset$-E) application or critical segment, the marking of the induction hypothesis by $Mark$, $[A(x)]$, is I.
(3) Otherwise, the marking of $[A(x)]$ is a proper superset of I.

According to the theorem, the declaration of the conclusion is as follows:
Case 1: If the proof tree of the induction step has the critical ($\supset$-E) application in one of the main paths from the induction hypothesis, the declaration must be trivial.
Case 2: If the proof tree of the induction step has no critical ($\supset$-E) application or critical segment, the declaration may be arbitrary.
Case 3: If the proof tree of the induction step has no critical ($\supset$-E) application but has critical segments, the declaration must be enlarged to eliminate critical segments. In this case, the marking of the induction hypothesis, S, and the initial declaration are different, so that the declaration should be same as S and the marking be performed again.

7. Program Extraction Algorithm

The proof compilation should be modified to handle marked proof trees. The chief modifications are:
1) if the given formula, A, is marked by $\{i_0, \cdots, i_k\}$, extract the code for the i_lth ($0 \leq l \leq k$) realizing variable in $Rv(A)$;
2) if formula A is marked by ϕ, no code should be extracted and there is no need to analyse the subtree determined by A;
3) if formula A is trivially marked, all the codes for $Rv(A)$ should be extracted.

The following is part of the definition of the modified version of the Ext procedure. $NExt$. $proj(I)(\bar{t})$ for a finite set of natural numbers, I, and a sequence of terms, $\bar{t}$, is the set of i-th projection ($i \in I$) of $\bar{t}$.

- $$NExt\left(\dfrac{\dfrac{\Sigma_0}{\{A_0\}_{I_0}} \cdots \dfrac{\Sigma_{n-1}}{\{A_{n-1}\}_{I_{n-1}}}}{\{A_0 \wedge \cdots \wedge A_{n-1}\}_I}(\wedge\text{-}I)\right) \stackrel{\text{def}}{=} \left(NExt\left(\dfrac{\Sigma_0}{\{A_0\}_{I_0}}\right), \cdots NExt\left(\dfrac{\Sigma_{n-1}}{\{A_{n-1}\}_{I_{n-1}}}\right)\right)$$

Note that if $I_i = \phi$, $NExt\left(\dfrac{\Sigma_i}{\{A_i\}_{I_i}}\right) = (nil) \quad i = 0 \cdots 1$.

- $$NExt\left(\dfrac{\dfrac{\Sigma}{\{A_0 \wedge \cdots \wedge A_{n-1}\}_I}}{\{A_i\}_I}(\wedge\text{-}E)\right) \stackrel{\text{def}}{=} NExt\left(\dfrac{\Sigma}{\{A_0 \wedge \cdots \wedge A_{n-1}\}_J}\right) \qquad \text{where } i = 0 \cdots n-1.$$

- $$NExt\left(\dfrac{\dfrac{\Sigma}{\{A\}_J}}{\{A \vee B\}_I}(\vee\text{-}I)\right) \stackrel{\text{def}}{=} \begin{cases} \left(left, NExt\left(\dfrac{\Sigma}{\{A\}_J}\right), any[k]\right) & \text{if } 0 \in I \\ \left(NExt\left(\dfrac{\Sigma}{\{A\}_J}\right), any[l]\right) & \text{if } 0 \notin I \end{cases}$$

- $$NExt\left(\cfrac{\cfrac{\Sigma}{\{B\}_J}}{\{A\vee B\}_I}(\vee\text{-}I)\right)\overset{\text{def}}{=}\begin{cases}(right, any[k], NExt\left(\cfrac{\Sigma}{\{B\}_J}\right)) & \text{if } 0\in I\\ (any[l], NExt\left(\cfrac{\Sigma}{\{B\}_J}\right)) & \text{if } 0\notin I\end{cases}$$

where $k =| I | -(1+ | J |)$ and $l =| I | - | J |$, and $any[n]$ is the sequence of any n codes.

- $$NExt\left(\cfrac{\cfrac{\Sigma_0}{\{A\vee B\}_{J_0}}\quad \cfrac{\overset{\{[A]\}_{J_1}}{\Sigma_1}}{\{C\}_I}\quad \cfrac{\overset{\{[B]\}_{J_2}}{\Sigma_2}}{\{C\}_I}}{\{C\}_I}(\vee\text{-}E)\right)$$

$$\overset{\text{def}}{=} if\ left = proj(\{0\})\left(NExt\left(\cfrac{\Sigma_0}{\{A\vee B\}_{J_0}}\right)\right)\ then\ NExt\left(\cfrac{\overset{\{[A]\}_{J_1}}{\Sigma_1}}{\{C\}_I}\right)\theta\ else\ NExt\left(\cfrac{\overset{\{[B]\}_{J_2}}{\Sigma_2}}{\{C\}_I}\right)\theta$$

otherwise

$$\text{where } \theta \overset{\text{def}}{=} \left\{\begin{array}{c} proj(J_1)(Rv(A))/proj(J_0+1)\left(NExt\left(\cfrac{\Sigma_0}{\{A\vee B\}_{J_0}}\right)\right), \\ proj(J_2)(Rv(B))/proj(J_1+(1+\mid J_0\mid))\left(NExt\left(\cfrac{\Sigma_0}{\{A\vee B\}_{J_0}}\right)\right)\end{array}\right\}$$

- $$NExt\left(\cfrac{\cfrac{\overset{[x:\sigma]}{\Sigma}}{\{A(x)\}_I}}{\{\forall x:\sigma.\ A(x)\}_I}(\forall\text{-}I)\right)\overset{\text{def}}{=}\lambda x.\ NExt\left(\cfrac{\overset{[x:\sigma]}{\Sigma}}{\{A(x)\}_I}\right)$$

- $$NExt\left(\cfrac{\cfrac{\overset{\{[A]\}_J}{\Sigma}}{\{B\}_I}}{\{A\supset B\}_I}(\supset\text{-}I)\right)\overset{\text{def}}{=}\lambda\ proj(J)(Rv(A)).\ NExt\left(\cfrac{\overset{\{[A]\}_J}{\Sigma}}{\{B\}_I}\right)$$

J is the union of all the markings of A used as assumptions.

- $$NExt\left(\cfrac{\cfrac{\Sigma_0}{A}\quad\cfrac{\Sigma_1}{\{A\supset B\}_I}}{\{B\}_I}(\supset\text{-}E)\right)\overset{\text{def}}{=}NExt\left(\cfrac{\Sigma_1}{\{A\supset B\}_I}\right)\left(NExt\left(\cfrac{\Sigma_0}{A}\right)\right)$$

- $$NExt\left(\cfrac{\cfrac{}{\{t:\sigma\}_J}(*)\quad\cfrac{\Sigma}{\{A(t)\}_K}}{\{\exists x:\sigma.\ A(x)\}_I}(\exists\text{-}I)\right)\overset{\text{def}}{=}\begin{cases}(t, NExt\left(\cfrac{\Sigma}{\{A(t)\}_K}\right)) & \text{if } J=\{0\}\\ NExt\left(\cfrac{\Sigma}{\{A(t)\}_K}\right) & \text{if } J=\phi\end{cases}$$

- $$NExt\left(\cfrac{\cfrac{}{\{\exists x:\sigma.\ A(x)\}_J}(*)\qquad\cfrac{\overset{[\{t:\sigma\}_K,\{A(t)\}_L]}{\Sigma}}{\{C\}_I}}{\{C\}_I}(\exists\text{-}E)\right)\overset{\text{def}}{=}NExt\left(\cfrac{\overset{[\{t:\sigma\}_K,\{A(t)\}_L]}{\Sigma}}{\{C\}_I}\right)\theta$$

$$\text{where } \theta \overset{\text{def}}{=} \left\{\begin{array}{c} proj(L)(Rv(A(x)))/proj(J(\geq 1))\left(NExt\left(\cfrac{}{\{\exists x:\sigma.\ A(x)\}_J}(*)\right)\right), \\ t/proj(\{0\})\left(NExt\left(\cfrac{}{\{\exists x:\sigma. A(x)\}_J}\right)\right)\end{array}\right\}.$$

where $I(\geq n) \overset{\text{def}}{=} \{x \in I \mid x \geq n\}$.

- $$NExt\left(\dfrac{\dfrac{\Sigma_0}{\{A(0)\}_I} \quad \dfrac{\overset{[x:nat,\,\{A(x)\}_I]}{\Sigma_1}}{\{A(succ(x))\}_I}}{\{\forall x : nat.\ A(x)\}_I}(nat\text{-}ind)\right)$$

$$\overset{\text{def}}{=} \mu\ \Xi.\ \lambda\ x.\ if\ x = 0\ then\ NExt\left(\frac{\Sigma_0}{\{A(0)\}_I}\right)\ else\ NExt\left(\frac{\overset{[x:nat,\,\{A(x)\}_I]}{\Sigma_1}}{\{A(succ(x))\}_I}\right)\sigma$$

$$\text{where } \Xi = proj(I)(Rv(A(x))), \text{and } \sigma = \{\Xi/\Xi(pred(x)), x/pred(x)\}$$

The following theorem shows that $Mark$ and $NExt$ can be seen as an extension of the projection function on the extracted codes.

Theorem 2: Soundness of the $NExt$ procedure
Let A be a sentence and D be the declaration. If $\vdash_{\mathrm{QPC}} A$ and Π is its proof tree, then

$$proj(D)(Ext(\Pi)) = NExt(Mark(\Pi))$$

8. Conclusion

A technique to eliminate the redundancy in realiser codes, the extended projection method, was presented in this paper. It performs program analysis in the form of proof theoretic analysis of proof trees.

REFERENCES

[Bates 79] Bates, J.L., "*A logic for correct program development*", Ph.D. Thesis, Cornell University, 1979

[Constable 86] Constable, R.L., "*Implementing Mathematics with the Nuprl Proof Development System*", Prentice-Hall, 1986

[Coquand 86] Coquand, T. and Huet, G., "*The Calculus of Constructions*", Rapports de Recherche N° 530, INRIA, 1986

[Hayashi 87] Hayashi, S. and Nakano, H., "*PX: a computational logic*", RIMS-573, RIMS, Kyoto University, 1987

[Mohring 88] Mohring-Paulin, C., 1988, personal communication

[Prawitz 65] Prawitz, D., "*Natural Deduction*", Almquist and Wiksell, Stockholm, 1965

[Sasaki 86] Sasaki, J., "*Extracting Efficient Code From Constructive Proofs*", Ph.D. Thesis, Cornell University, 1986

[Sato 85] Sato, M., "*Typed Logical Calculus*", Technical Report 85-13, Department of Information Science, Faculty of Science, University of Tokyo, 1985

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

[Takayama 88a] Takayama, Y., "QPC: QJ-based Proof Compiler –Simple Examples and Analysis", *Proceeding of European Symposium on Programming '88* LNCS Vol. 300, Springer-Verlag, 1988, also published as Technical Report TR-296, ICOT

[Takayama 88b] Takayama, Y., "Proof Theoretic Approach to the Extraction of Redundancy-free Realiser Code", Technical Report, ICOT, 1988, (to appear)