TR-522

Extraction of Redundancy-free Programs from Constructive Natural Deduction Proofs

by
Y. Takayama

November, 1989

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

Submitted for publication in J. Symbolic Computation

Extraction of Redundancy-free Programs from Constructive Natural Deduction Proofs

Yukihide Takayama

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

Abstract

Executable codes can be extracted from constructive proofs by using realizability interpretation. However, realizability also generates redundant codes that have no significant computational meaning. This redundancy causes heavy runtime overhead, and is one of the obstacles in applying realizability to practical systems that realize the mathematical programming paradigm. This paper presents a method to eliminate redundancy by analyzing proof trees as pre-processing of realizability 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 realizer is needed. This procedure does not always work well. This paper also gives an analysis of it and techniques to resolve critical cases. The method is studied in a simple constructive logic with primitive types, mathematical induction and its **q**-realizability interpretation. As an example, the extraction of a prime number checker program is given.

Keywords: constructive logic, realizability, natural deduction, proof tree analysis, proof compilation

1. Introduction

Writing programs as constructive proofs of theorems is thought to be one good approach to automated programming and program verification [Constable 86] [Takayama 87]. Executable codes can be extracted from constructive proofs by using the Curry-Howard isomorphism of formulas-as-types [Howard 80], or the notion of realizability [Troelstra 73]. Here, it raises the problem of extracting efficient codes from proofs, or, in other words, optimization at proof level.

[Bates 79] applied a traditional syntactical optimization technique to the code extracted from proofs which is a set of source-to-source transformation rules. A technique to optimize programs at proof level, *pruning*, is given in [Goad 80]. Generally, a proof contains a lot of information about the program that corresponds to the proof, and the pruning technique uses the information in optimization that drastically changes the strategies of algorithms. [Sasaki 86] introduced the technique called *singleton justification* to Bates' program extraction algorithm so that the trivial codes for formulas that have no computational meaning can be simplified. The basic idea is as follows: if A and B are atomic formulas, then the computational meaning is trivial, so that the code extracted from, for instance, $A \wedge B$, which is called *singleton formula*, is $(trivial, trivial)$. The modified program extractor simplifies the code to $trivial$. The class of the singleton formulas is essentially equal to that of Harrop formulas [Troelstra 73]. The **QPC** system [Takayama 88] uses a similar technique to singleton justification, proof normalization method to eliminate β-redex in the extracted codes, the *modified* $\vee$ *code* technique to simplify some classes of decision procedures and a few other code simplification rules as in Bates' and Sasaki's program extractors. However, the code extracted from constructive proofs still has redundancy, the redundant verification code, and it causes heavy runtime overhead.

The formalization of the problem in this paper is as follows. If, for example, a constructive proof of the following formal specification is given:

$$\forall x : \sigma_0.\ \exists y : \sigma_1.\ A(x, y)$$

where σ_0 and σ_1 are types, and $A(x, y)$ is a formula with free variables, x and y, a function, f, which satisfies the following condition can be extracted by **q**-realizability:

$$\forall x : \sigma_0.A(x, f(x)).$$

For example, if the proof is as follows:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}[x : \sigma_0]\\ \Sigma_0\end{array}}{t_x : \sigma_1} \quad \dfrac{\begin{array}{c}[x : \sigma_0]\\ \Sigma_1\end{array}}{A(x, t_x)}}{\exists y : \sigma_1.\ A(x, y)}(\exists\text{-}I)}{\forall x : \sigma_0.\ \exists y : \sigma_1.\ A(x, y)}(\forall\text{-}I)$$

where Σ_0 and Σ_1 denote sequences of subtrees, the extracted code can be expressed as:

$$\lambda x.\ (t_x, T)$$

where T is the code extracted from the subtree, $(\Sigma_1 / A(x, t_x))$, t_x denotes a term which contains a free variable, x, and $(,)$ is the sequence constructor. In this paper, the executable code extracted from a constructive proof, which is called *realizer code* or simply *realizer* is in the form of sequence of terms or a function which outputs a sequence of terms. The code contains verification information which is not necessary in practical computation. In this case, the expected code is:

$$f \stackrel{\text{def}}{=} \lambda x.\ t_x$$

so that T is the redundant code.

The most reasonable way to overcome this problem would be to introduce 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] and ITT implemented by the Gödeborg group [Nordström 83] as a weaker notion of $\exists x : A.\ B$. This is used to skip the extraction of the justification for B. [Paulin-Mohring 89] modified the Calculus of Constructions [Coquand 88] by introducing two kinds of constants, *Prop* and *Spec*, to distinguish the formulas, in proofs, whose computational meaning is not necessary. These works are performed in the type theoretic formulation of constructive logic in the style of Martin-Löf or higher order λ-calculus with dependent types. For the non-type theoretic formulation of constructive logic, $\Diamond$-bounded formulas introduced in PX [Hayashi 88] play a similar role to the set notation from which no realizer code is extracted.

This paper presents another method for the program analysis at proof tree level, and for extraction of a redundancy-free realizer code in a non-type theoretic formalization of constructive logic. In some cases, the redundancy can be removed easily by applying a simple operation to the extracted code. For example, if the 0th element of the realizer code, $(t_0, t_1, \cdots, t_n)$, from a proof of $\exists x : \sigma.\ A(x)$ is needed, it is obtained by applying projection function: $proj(0)(t_0, t_1, \cdots, t_n) = t_0$. If the theorem is in the form of $\forall x : \sigma.\ A(x)$, the realizer from its proof is in the form of $\lambda x.(t_0, \cdots, t_n)$, so the procedure is a little more complicated: Translate the realizer to $(\lambda x.t_0, \cdots, \lambda x.t_n)$ and apply projection function. However, the situation around the redundancy is more complicated when the program extraction is performed on proofs in induction, in other words, when recursive call programs are extracted. It needs rather sophisticated program analysis. For example, assume that the following recursive call program is extracted from a proof in induction:

$$\mu(z_0, z_1).\ \lambda x.\ if\ x = 0\ then\ (0, 1)\ else\ (f_{z_0, z_1}, g_{z_0, z_1})$$

where μ is a fixed point operator and both of the parameters, z_0 and z_1, of μ actually occur in f. This function calculates a sequence of terms of length 2, and both of the elements of the sequence calculated at the recursive call step are necessary to calculate the 0th element of the sequence. Therefore, it is impossible to extract only the 0th element of the realizer code. The program analysis of redundancy 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 them.

Section 2 defines a simple constructive logic used in this paper. This is basically an intuitionistic first order natural deduction with mathematical induction and a variant of type-free λ-calculus. The program extraction algorithm, Ext, is defined here. Ext performs q-realizability interpretation of proofs. Section 3 introduces the notion of *marking* which is a basic tool for the analysis of proof trees. The marking procedure may fail if the proof uses mathematical induction. This is explained in Section 4. The modified proof extraction algorithm, $NExt$, is defined in Section 5. $NExt$ generates redundancy-free codes from the proof tree analyzed by the marking procedure. Section 6 gives a few properties and characterization of marking and $NExt$. A prime number checker program is investigated as an example in Section 7. Section 8 gives some discussion and the concluding remark.

2. Simple Constructive Logic

The constructive logic used here is an intuitionistic version of first order natural deduction with mathematical induction. It has a type-free λ-calculus as terms, and equality and inequality between terms. It is a sugared subset of Sato's theory, **QJ** [Sato 85] [Sato 86].

2.1 Expressions and Inference Rules

Only the part of the definitions which is sufficient to enable understanding of the contents of succeeding sections will be given. See [Takayama 88] for details.

Types are used as domains of quantified variables.

Definition 1: *Types*
The primitive types are *nat*, **2**, and *bool*. A type is constructed with primitive types and type constructors, → (function type constructor) and × (cartesian product constructor).

Definition 2: *Substitutions*
A substitution is denoted $\{X_0/T_0, \cdots, X_{n-1}/T_{n-1}\}$ which means substituting T_i for X_i, and X_i is a variable or a sequence of variables. If X_i is a sequence of variables, T_i must be a sequence of terms. Application of a substitution, θ, to a term, T, is denoted $T\theta$.

Definition 3: *Terms (program constructs)*

1) Atoms:
   - Elements of *nat*: $0, 1, 2, \cdots$
   - Elements of **2**: *left* and *right*
   - Elements of *bool*: T and F
2) Variables: $x, y, z, \cdots$
3) Sequence:
   If $t_0, \cdots, t_n$ are terms, then sequence of the terms, $(t_0, \cdots, t_n)$ is also a term.
   A sequence of variables, $(x_0, \cdots, x_{n-1})$, will often be denoted $\overline{x}$. Nil sequence is denoted $()$. $any[n]$ $(n \geq 0)$ denotes a sequence of any atoms, and $any[0] = ()$
4) Abstraction:
   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 $M(N)$, or simply $M\ N$, is a term;
6) If-then-else:
   If A is an equation or inequation of terms and *beval* is a function which determines whether A is true or not and returns boolean values, and if S_0 and S_1 are terms, then *if* $beval(A)$ *then* S_1 *else* S_2 is a term. $beval(A)$ is often abbreviated to A;
7) Fixed point:
   If M is a term which contains the variables, $z_0, \cdots, z_n$, free, then $\mu(z_0, \cdots, z_n).\ M$ is a term;

8) Built-in functions:

- $succ$, $pred$, $+$, $-$, $/$ $\cdot$ • $proj(n)$ $\cdots$ n-th projection function;
- $proj(I)$ where I is a finite sequence of natural numbers. In the following, M and N are terms or sequences of terms and X is a variable or a sequence of variables.

  a) $proj(\{i_0, \cdots, i_m\})\ (S) \overset{\text{def}}{=} (proj(i_0)(S), \cdots, proj(i_m)(S))$
  where S is a sequence of terms of length n $(m < n)$;

  b) $proj(I)(\lambda X.\ M) \overset{\text{def}}{=} \lambda X.\ proj(I)(M)$

  c) $proj(I)(any[n]) \overset{\text{def}}{=} any[k] \qquad (k = \text{length of } I) \leq n)$

  d) $proj(I)(if\ beval(A)\ then\ M\ else\ N)$
  $\overset{\text{def}}{=} if\ beval(A)\ then\ proj(I)(M)\ else\ proj(I)(N)$

  e) $proj(I)(M\theta) = (proj(I)(M))\theta$ where θ is a substitution;

  f) $proj(I)(M(N)) \overset{\text{def}}{=} (proj(I)(M))(N)$

- For a sequence of terms, S, of length n,
  $tseq(i)(S) \overset{\text{def}}{=} (proj(i)(S), proj(i+1)(S), \cdots, proj(n-1)(S))$
  where $0 \leq i \leq n-1$
  $ttseq(i,l)(S) \overset{\text{def}}{=} (proj(i)(S), proj(i+1)(S), \cdots, proj(i+(l-1))(S))$
  where $0 \leq i \leq n-1, 1 \leq l \leq n-i$

Sequence of terms:
If $S_1, \cdots, S_2$ are sequences of terms, then their concatenation is denoted $(S_1, \cdots, S_2)$. $(S, ())$ and $((), S)$ are equal to S. Also, the following equivalence relations are given:

- $if\ beval(A)\ then\ (M_0, \cdots, M_n)\ else\ (N_0, \cdots, N_n)$
  $\equiv (if\ beval(A)\ then\ M_0\ else\ N_0, \cdots, if\ beval(A)\ then\ M_n\ else\ N_n)$
- $\lambda X.\ (M_0, \cdots, M_n) \equiv (\lambda X.M_0, \cdots, \lambda X.M_n)$
- $(M_0, \cdots, M_n)(N) \equiv (M_0(N), \cdots, M_n(N))$

Fixed point:
A fixed point used here has a sequence of variables as parameters. A fixed point

$$\mu(z_0, \cdots, z_{n-1}).\ M$$

denotes a solution of the following fixed point equation:

$$(z_0, \cdots, z_{n-1}) = M$$

If the term M is equivalent to a sequence of terms of length n,

$$M \equiv (M_0, \cdots, M_{n-1}) \qquad (*)$$

the fixed point equation can be solved, and the solution, $(f_0, \cdots, f_{n-1})$ is as follows:

$$f_i = \mu z_i.\ M_i\{z_0/f_0, \cdots, z_{i-1}/f_{i-1}, z_{i+1}/f_{i+1}, \cdots, z_{n-1}/f_{n-1}\} \qquad (0 \leq i \leq n-1)$$

Therefore, the following equivalence relation is introduced:

$$\mu(z_0, \cdots, z_{n-1}).\ M \equiv (f_0, \cdots, f_{n-1}) \qquad \text{if } (*) \text{ holds}$$

Definition 4: *Formula*
1) $\perp$ is an atomic formula;
2) Equation and inequation of terms are atomic formulas;
3) If M is a term and σ is a type, then $M : \sigma$ is an atomic formula;
4) If A and B are formulas, then $A \wedge B$, $A \vee B$ and $A \supset B$ are formulas;
5) If x is a variable, σ is a type and A is a formula, then $\exists x : \sigma.A$ and $\forall x : \sigma.A$ are formulas;

Negation of a formula, $\neg A$, is defined as $\neg A \overset{\text{def}}{=} A \supset \perp$. The type declarations of bound variables are often omitted. Also atomic formula $M : \sigma$ is often denoted simply M.

Inference rules are as follows:

- Introduction and elimination rules on $\wedge$, $\vee$, $\supset$, $\forall$ and $\exists$
- $\perp$ elimination rule
- Mathematical induction rule
- Rules on equality and inequality of terms
- Term construction rules

$*$ is used as the abbreviation of the names of equality rules, term construction rules, and axioms.

2.2 Proof Theoretic Terminology and Notation

This section gives basic proof theoretic terminologies used in the following description.

- Π always stands for proof trees, and Σ for sequences of proof trees.
- Assumptions discharged in the deduction are enclosed by square brackets: []. Note that this is different from Prawitz's notation, in which both parentheses and square brackets are used: () and [].

Definition 5: *Principal sign & C formula*
1) Let F be a formula that is not atomic. Then, F has one of the forms $A \wedge B$, $A \vee B$, $A \supset B$, $\forall x.A$, and $\exists x.A$; the symbol $\wedge$, $\vee$, $\supset$, $\forall$, or $\exists$ is called the ***principal sign*** of F.
2) A formula with the principal sign, C, is called the *C formula*.

Definition 6: *Application & node*
In a proof tree as follows

$$\cfrac{\cfrac{\Sigma_0}{A_0} \cdots \cfrac{\Sigma_n}{A_n}}{B}\,(R)$$
$$\Pi$$

the formula occurrences, $A_0, \cdots, A_n$ and B, are called *nodes*, and the $\dfrac{A_0 \ \cdots \ A_n}{B}(R)$ part is called *application* of rule R, or *R application.*

Definition 7: *Subtree*

If A is a formula occurrence in a proof tree Π, *the subtree of* Π *determined by* A is the proof tree obtained from Π by removing all formula occurrences except A and the ones above A.

When a proof tree

$$\frac{\dfrac{\Sigma_0}{B_0} \ \cdots \ \dfrac{\Sigma_i}{B_i} \ \cdots \ \dfrac{\Sigma_{n-1}}{B_{n-1}}}{C}$$

is given, the subtree determined by B_i will often be denoted as (Σ_i/B_i).

Definition 8: *Top- & end-formula*

1) A *top-formula* in a proof tree, Π, is a formula occurrence that does not stand immediately below any formula occurrence in Π.

2) An *end-formula* of Π is a formula occurrence in Π that does not stand immediately above any formula occurrence in Π.

Definition 9: *Side-connected*

Let A be a formula occurrence in Π, let $(\Pi_0, \Pi_1, \cdots, \Pi_{n-1}/A)$ be the subtree of Π determined by A, and let $A_0, A_1, \cdots, A_{n-1}$ be the end formulas of $\Pi_0, \Pi_1, \cdots, \Pi_{n-1}$. Then, A_i is said to be *side-connected* with A_j $(0 \leq i, j < n)$.

Definition 10: *Minor & major premise*

In the following rules, Cs as premises of the rules, C_0, and C_1 are said to be *minor premises.* A premise that is not minor is called a *major premise.*

$$\frac{C \supset B \quad C}{B}(\supset\text{-}E) \qquad\qquad \frac{\exists x.\ A(x) \quad \begin{matrix}[A(x)]\\ C\end{matrix}}{C}(\exists\text{-}E)$$

$$\frac{A \vee B \quad \begin{matrix}[A]\\ C_0\end{matrix} \quad \begin{matrix}[B]\\ C_1\end{matrix}}{C}(\vee\text{-}E) \qquad \text{where } C_0 = C_1 = C$$

C_0 is called *left minor premise*, and C_1 is called *right minor premise.*

2.3 Realizing Variable Sequences and Length of Formulas

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

Definition 11: $Rv(A)$

1) $Rv(A) \stackrel{\text{def}}{=} ()$, if A is atomic; 2) $Rv(A \wedge B) \stackrel{\text{def}}{=} (Rv(A), Rv(B))$;

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

4) $Rv(A \supset B) \stackrel{\text{def}}{=} Rv(B)$; 5) $Rv(\forall x.\ A(x)) \stackrel{\text{def}}{=} Rv(A(x))$;

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

Example 1: :

$Rv(\forall x : nat.\ (x \geq 0 \supset (x = 0 \vee \exists y : nat.\ succ(y) = x))) = (z_0, z_1)$

where z_0 denotes the information that shows which subformula of the $\vee$ formula holds and z_1 denotes the realizing variables of $\exists y : nat.\ succ(y) = x$. Note that $Rv(succ(y) = x) = ()$.

Definition 12: *Length of formulas*

$l(A)$, which is called the *length of formula* A, is the length of $Rv(A)$.

2.4 Proof Compilation (*Ext* Procedure)

The realizability used in this paper is a variant of **q**-realizability defined in [Sato 85]. The chief difference from the standard **q**-realizability as seen in Chapter VII of [Beeson 85] is that the realizer code for an atomic formula is defined as nil sequence here while there is no such restriction in the standard **q**-realizability. **px**-realizability [Hayashi 88] also has the same restriction. Another difference from the standard form is the definition of realizability of $\vee$ formulas. The standard **q**-realizability defines the realizer code of $A \vee B$ as $(left, \overline{a})$ or $(right, \overline{b})$ in which $left$ and $right$ are the flags to show which formula of the disjunction actually holds and $\overline{a}$ and $\overline{b}$ are realizer codes of A and B. On the other hand, it is defined as $(left, \overline{a}, any[l(B)])$ or $(right, any[l(A)], \overline{b})$ in this paper.

The realizability is reformulated here as the *Ext* procedure [Takayama 88] that takes proof trees as input and returns functional style programs as output.

(1) For the realizer code of an assumption, the realizing variable sequence is used:

$Ext(A) \stackrel{\text{def}}{=} Rv(A)$

(2) No significant code is extracted from an atomic formula:

$Ext\left(\frac{\Sigma}{A}(Rule)\right) \stackrel{\text{def}}{=} ()$ where A is an atomic formula.

(3) The realizer codes of $\wedge$ formulas and $\vee$ formulas are denoted as sequences. The constants $left$ and $right$ are used to denote the information indicating which of the formulas connected by $\vee$ actually holds.

- $Ext\left(\dfrac{\dfrac{\Sigma_0}{A_0} \quad \dfrac{\Sigma_1}{A_1}}{A_0 \wedge A_1}(\wedge\text{-}I)\right) \stackrel{\text{def}}{=} \left(Ext\left(\dfrac{\Sigma_0}{A_0}\right), Ext\left(\dfrac{\Sigma_1}{A_1}\right)\right)$

- $Ext\left(\dfrac{\dfrac{\Sigma}{A_0 \wedge A_1}}{A_i}(\wedge\text{-}E)_i\right) \qquad (i = 0, 1)$

$\stackrel{\text{def}}{=} ttseq(p,q)\left(Ext\left(\dfrac{\Sigma}{A_0 \wedge A_1}\right)\right) \quad \text{where } (p,q) = \begin{cases} (0, l(A_0)) & \text{if } i = 0 \\ (l(A_0), l(A_1)) & \text{if } i = 1 \end{cases}$

- $Ext\left(\dfrac{\dfrac{\Sigma}{A}}{A \vee B}(\vee\text{-}I)_0\right) \stackrel{\text{def}}{=} \left(left, Ext\left(\dfrac{\Sigma}{A}\right), any[l(B)]\right)$

- $Ext\left(\dfrac{\dfrac{\Sigma}{B}}{A \vee B}(\vee\text{-}I)_1\right) \stackrel{\text{def}}{=} \left(right, any[l(A)], Ext\left(\dfrac{\Sigma}{B}\right)\right)$

(4) The realizer code extracted from the proof in the ($\vee$-E) rule is the *if-then-else* program. If the decision procedure of $A \vee B$ is simple (directly executable on computers), Ext generates the *modified* $\vee$ *code* [Takayama 88].

$$Ext\left(\dfrac{\dfrac{\Sigma_0}{A \vee B} \quad \dfrac{\overset{[A]}{\Sigma_1}}{C} \quad \dfrac{\overset{[B]}{\Sigma_2}}{C}}{C}(\vee\text{-}E)\right)$$

is as follows:

a) $if\ beval(A)\ then\ Ext\,(\Sigma_1/C)\ else\ Ext\,(\Sigma_2/C)$ [modified $\vee$ code]
$\cdots$ when both A and B are equations or inequations of terms

b) $if\ left = proj(0)(Ext\,(\Sigma_0/A \vee B))\ then\ Ext\,(\Sigma_1/C)\,\theta\ else\ Ext\,(\Sigma_2/C)\,\theta$
$\cdots$ otherwise

where $\theta \stackrel{\text{def}}{=} \left\{\begin{array}{l} Rv(A)/ttseq(1, l(A))\,(Ext\,(\Sigma_0/A \vee B)), \\ Rv(B)/tseq(l(A)+1)\,(Ext\,(\Sigma_0/A \vee B)) \end{array}\right\}$

(5) λ expressions are extracted from the proofs in ($\supset$-I) and ($\forall$-I):

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

- $\displaystyle Ext\left(\frac{\begin{array}{c}[A]\\ \underline{\Sigma}\\ B\end{array}}{A\supset B}(\supset\text{-}I)\right) \stackrel{\text{def}}{=} \lambda Rv(A).\ Ext\left(\frac{\Sigma}{B}\right)$

(6) The code that is in the form of a function application is extracted from the proofs in $(\supset\text{-}E)$ and $(\forall\text{-}E)$:

- $\displaystyle Ext\left(\frac{\dfrac{\Sigma_0}{A\supset B}\quad \dfrac{\Sigma_1}{A}}{B}(\supset\text{-}E)\right) \stackrel{\text{def}}{=} Ext\left(\frac{\Sigma_0}{A\supset B}\right)\left(Ext\left(\frac{\Sigma_1}{A}\right)\right)$

- $\displaystyle Ext\left(\frac{\dfrac{\Sigma_0}{t:\sigma}\quad \dfrac{\Sigma_1}{\forall x:\sigma.\ A(x)}}{A(t)}(\forall\text{-}E)\right) \stackrel{\text{def}}{=} Ext\left(\frac{\Sigma_1}{\forall x:\sigma.\ A(x)}\right)(t)$

(7) The codes extracted from proofs in $(\exists\text{-}I)$ and $(\exists\text{-}E)$ are as follows:

- $\displaystyle Ext\left(\frac{\dfrac{\Sigma_0}{t:\sigma}\quad \dfrac{\Sigma_1}{A(t)}}{\exists x:\sigma.\ A(x)}(\exists\text{-}I)\right) \stackrel{\text{def}}{=} \left(t, Ext\left(\frac{\Sigma_1}{A(t)}\right)\right)$

- $\displaystyle Ext\left(\frac{\dfrac{\Sigma_0}{\exists x:\sigma.\ A(x)}\qquad \begin{array}{c}[x:\sigma, A(x)]\\ \underline{\Sigma_1}\\ C\end{array}}{C}(\exists\text{-}E)\right) \stackrel{\text{def}}{=} Ext\left(\frac{\Sigma_1}{C}\right)\theta$

where

$$\theta \stackrel{\text{def}}{=} \{Rv(A(x))/tseq(1)(Ext(\Sigma_0/\exists x:\sigma.\ A(x))), x/proj(0)(Ext(\Sigma_0/\exists x:\sigma.\ A(x)))\}.$$

(8) Any code is extracted from a proof in the $(\bot\text{-}E)$ rule:

- $\displaystyle Ext\left(\frac{\begin{array}{c}\underline{\Sigma}\\ \bot\end{array}}{A}(\bot\text{-}E)\right) \stackrel{\text{def}}{=} any[l(A)].$

(9) The code extracted from $(=\text{-}E)$ rule is as follows:

- $\displaystyle Ext\left(\frac{\dfrac{\Sigma_0}{x=y}\quad \dfrac{\Sigma_1}{A(x)}}{A(y)}(=\text{-}E)\right) \stackrel{\text{def}}{=} Ext\left(\frac{\Sigma_1}{A(x)}\right)$

(10) Multi-valued recursive call functions are extracted from the proofs in mathematical induction.

$$\bullet\ Ext\left(\dfrac{\dfrac{\Sigma_0}{A(0)} \qquad \dfrac{\begin{matrix}[x : nat, x > 0, A(pred(x))]\\ \Sigma_1\end{matrix}}{A(x)}}{\forall x : nat.\ A(x)}(nat\text{-}ind)\right)$$

$\stackrel{\rm def}{=} \mu\,\overline{z}.\ \lambda\,x.\ if\ x = 0\ then\ Ext\,(\Sigma_0/A(0))\ \ else\ Ext\,(\Sigma_1/A(x))\,\sigma$

where $\overline{z}$ is a sequence of new variables whose length is $l(A(pred(x)))$, and
$\sigma = \{Rv(A(pred(x)))/\overline{z}(pred(x))\}$.

Theorem 1: (*Soundness of the Ext procedure*)
Let A be a formula. If Π_A is a proof of A, then $\vdash Ext(\Pi_A)\ \mathbf{q}\ A$ where $a\ \mathbf{q}\ A$ means that a term, a, realizes the formula A, and $FV(A) \supset FV(Ext(\Pi_A))$

Proof: By straightforward conversion from the proof of the theorem on the soundness of realizability interpretation of **QJ**. See [Sato 85]. ∎

Lemma 1: *Let A and Π_A be a formula and its proof. Then the code, $Ext(\Pi_A)$, is equivalent to a sequence of terms of length $l(A)$*

Proof: Induction on the construction of Π_A. The crucial point is that if A is a $\forall$ formula, $\forall x.\ B(x)$, and proved in mathematical induction, and if $Ext(\Pi_A)$ is $\mu(z_0, \cdots, z_{n-1}).\ M$, then M is equivalent to a sequence of terms of length $n = l(B(pred(x))) = l(B(0)) = l(B(x))$. Then, $Ext(\Pi_A)$ is equivalent to a sequence of terms as explained in 2.1. ∎

The realizer code extracted by *Ext* is equivalent to a sequence of terms, so that a realizer will also be called a *realizer sequence*.

3. Declaration and Marking of Proof Trees

The proof trees are a clear description of the logical meaning of programs, so that analysis to detect the redundancy of realizer codes is much easier if it is performed at the proof tree level.

The realizer of a formula, A, is a sequence of terms of length $l(A)$ according to lemma 1 in the last section. However, not all the elements of the sequence are always necessary. In addition, it is generally difficult to determine automatically which part of the realizer code is really necessary, so end users must specify which elements of the realizer codes of each node are needed, but at the same time it is preferable to limit the information that end users must specify. The basic requirement is that end users should not need to understand how the proof compiler works in order to specify the redundant part of the proof in terms of computation.

On the other hand, the proof compiler performs realizability interpretation. It analyses a given proof tree from bottom to top, extracting the code step by step for the inference rule of each application in the proof tree, so that, if the path of the proof tree analysis by the proof compiler is traced, the information given to the end-formula can be propagated from bottom to top of the proof tree being reformed according to the inference rule of each application. The proof compiler uses the information to refrain from generating unnecessary code. Consequently, end users need not specify the information about redundancy at all the nodes in the proof tree; it is enough to specify them only at the conclusion of the proof.

3.1 Declaration to Specifications

Definition 13: *Declaration*

(1) *Declaration*, I, of a specification, A, is a subset of the finite set of natural numbers, $\{0, 1, \cdots, l(A)-1\}$. I is always assumed to be sorted: Assume $I = \{i_1, \cdots, i_n\}$, then $i_p < i_q$ if $p < q$. Therefore, I is also regarded as a sorted sequence of natural numbers.

A specification, A, with the declaration, I, is denoted $\{A\}_I$ or simply A_I.

Elements of the declaration are called *marking numbers*.

(2) The empty set, ϕ, is called *nil declaration*.

(3) The declaration, $\{0, 1, \cdots, l(A) - 1\}$, is called *trivial*, and denoted TRV.

In the following, a declaration to the conclusion of a proof, Π, will often be called a ***declaration to a proof***, Π, or ***a declaration given to*** Π. A declaration is a set of the position numbers of the realizer sequence that specifies which elements of the realizer sequence are needed. It is the only information that end users of the system need to specify: the other part is performed automatically.

Example 2: Let $\forall x_0. \cdots \forall x_{m-1}. \exists y_0. \cdots \exists y_{n-1}.\ A(x_0, \cdots, x_{m-1}, y_0, \cdots, y_{n-1})$ be the specification and assume that 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: $\{0, \cdots, k\}$

Example 3: $A \overset{\text{def}}{=} \forall x.\ (x \geq 3 \supset \forall y. \exists z. \exists w.\ x = y * z + w) \wedge (0 \leq w < y)$ is a specification of division of natural numbers more than 3. $Rv(A) = (z_0, z_1)$, where z_0 corresponds to $\exists z$ and z_1 to $\exists w$. In other words, a realizer of A is the sequence of a value of z and a value of w. If the function that calculates the remainder of division of x by y is needed, the declaration of A is $\{1\}$.

Example 4: $B \overset{\text{def}}{=} \forall x.(\exists y.x = 2 \cdot y) \vee (\exists z.x = 2 \cdot y + 1)$ is a specification of the program which checks whether the given natural number, x, is even or odd. The program extracted by Ext from a proof of B calculates the triples $(left, V_y, any[1])$, if x is even, and $(right, any[1], V_z)$.

if x is odd, in which V_y and V_z are the values of y and z. The constants, $left$ and $right$, indicate whether x is prime or not. Therefore, the declaration should be $\{0\}$ to generate redundancy-free program.

3.2 Marking

Definition 14: *Marking*

(1) *Marking*, I, of a node A in a proof tree is $\{0\}$ or ϕ if A is in the form of $M : \sigma$. Otherwise, marking of the node is a subset of the finite set of natural numbers, $\{0, 1, \cdots, l(A)-1\}$. As in the definition of declaration, I is also regarded as a sorted sequence of natural numbers. A node, A, with the marking, I, is denoted $\{A\}_I$ or simply A_I.
Elements of a 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*, and denoted TRV.

Note that declaration is a special case of marking; the marking of the end-formula of a proof tree is called declaration. A marking of the conclusion of a subtree, Π, of a tree will often be called a *marking of* Π, or a *marking given to* Π.

The marking procedure means to attach to each node of given proof trees the information that indicates which codes among the realizer sequence of the node are needed. The marking can be determined according to the inference rule of each node and the declaration. Let, for example, $\forall x.\ \exists y.\ \exists z.\ A(x,y,z)$ be the specification of a program and a function from x to y and z is the expected code from the proof of this specification. Let the proof be as follows:

$$\dfrac{\dfrac{}{s}(*) \qquad \dfrac{\dfrac{}{t}(*) \qquad \dfrac{\begin{matrix}[x]\\ \Sigma\end{matrix}}{A(x,s,t)}}{\exists z.\ A(x,s,z)}(\exists\text{-}I)}{\dfrac{\exists y.\ \exists z.\ A(x,y,z)}{\forall x.\ \exists y.\ \exists z.\ A(x,y,z)}(\forall\text{-}I)}(\exists\text{-}I)$$

The code extracted by Ext is

$$\lambda x.\ (s, t, Ext(\Sigma/A(x,s,t)) \equiv (\lambda x.s,\ \lambda x.t,\ \lambda x.Ext(\Sigma/A(x,s,t))).$$

However, only the 0th and 1st codes are needed here, so that the declaration is $\{0, 1\}$. The marking of $\exists y.\exists z.\ A(x,y,z)$, $\{0,1\}$, is determined according to the inference rule ($\forall$-I) and the declaration. For the node, $\exists z.\ A(x,s,z)$, the 0th code of the realizer sequence is the 1st code of $\exists y.\exists z.A(x,y,z)$, so that the marking is $\{0\}$. For $A(x,s,t)$, no realizer code is necessary here so that the marking is ϕ. t and s should also be marked by $\{0\}$, which indicates that s and t themselves are necessary. Consequently, the following tree is obtained:

$$\dfrac{\dfrac{\dfrac{}{\{s\}_{\{0\}}}(*) \qquad \dfrac{\dfrac{}{\{t\}_{\{0\}}}(*) \qquad \dfrac{\begin{matrix}\{[x]\}_{\phi}\\ \Sigma\end{matrix}}{\{A(x,s,t)\}_{\phi}}}{\{\exists z.\ A(x,s,z)\}_{\{0\}}}(\exists\text{-}I)}{\{\exists y.\ \exists z.\ A(x,y,z)\}_{\{0,1\}}}(\exists\text{-}I)}{\{\forall x.\ \exists y.\ \exists z.\ A(x,y,z)\}_{\{0,1\}}}(\forall\text{-}I)$$

Definition 15: *Marked proof tree*
The *marked proof tree* is a tree obtained from a proof tree and the declaration by the marking procedure.

The proof compilation procedure, Ext, should be modified to take marked proof trees as inputs and extract part of the realizer code according to the marking. It will be defined later. The formal definition of the marking procedure, called $Mark$, will also be given later, but before that, part of the definition will be given rather informally to make the idea clearer.

3.2.1 Marking of the ($\exists$-I) rule

By definition, the 0th code of

$$Ext\left(\dfrac{\dfrac{\Sigma_0}{t} \quad \dfrac{\Sigma_1}{A(t)}}{\exists x.A(x)}(\exists\text{-}I)\right)$$

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 the 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, K, of natural numbers, $K - 1 \overset{\text{def}}{=} \{a - 1 | a \in K, a - 1 \geq 0\}$.

3.2.2 Marking of the ($\exists$-E) Rule

By the definition of the Ext procedure, the realizer code of C concluded by the following inference is obtained by instantiating the code from the subtree determined by the minor premise by the code from the subtree determined by the major premise:

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

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, and let J and K be the unions of the markings of all occurrences of the two hypotheses, x and $A(x)$. Note that J is either $\{0\}$ or ϕ.

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

The marking of the major premise, $\exists x.A(x)$, is as follows:

Case 1: $J = \{0\}$

This means that the following reasoning is contained in the subtree determined by the minor premise in which x occurs in s:

$$\dfrac{\dfrac{\begin{matrix}[x] \\ \Sigma_2\end{matrix}}{s_x} \quad \dfrac{\Sigma_3}{P(s)}}{\exists y.\ P(y)}\,(\exists\text{-}I) \qquad\qquad \dfrac{\dfrac{\begin{matrix}[x] \\ \Sigma_4\end{matrix}}{s_x} \quad \dfrac{\Sigma_5}{\forall y.\ P(y)}}{P(s)}\,(\forall\text{-}E)$$

and the union of the marking of all the occurrences of x in Σ_2 or Σ_4 is $\{0\}$ so that the value of x should be extracted from the proof tree determined by the major premise. Consequently, the 0th element of the sequence of realizer 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 $\{0\} \cup (K+1)$ where $K + 1 \overset{\text{def}}{=} \{a+1 | a \in K\}$.

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 $K + 1$.

3.2.3 Marking of the ($\supset$-E) Rule

Let I be the marking of the conclusion, B, of a ($\supset$-E) application. The realizer of $A \supset B$ is a function that takes a realizer of A as input and returns a realizer of B. Then, I specifies the part of the output of the realizer of $A \supset B$ that is needed, so that $A \supset B$ should also be marked by I.

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

The marking of A should be TRV. The reason is as follows: The code extracted from (Σ_0/A) is the input of the function extracted from $(\Sigma_1/A \supset B)$. However, the marking, I, of $A \supset B$ is only to restrict the output of the function.

The marking of the subtree determined by A is continued recursively.

3.2.4 Definition of the *Mark* Procedure

- Notational preliminary

Mark is defined in the following style:

$$Mark\left(\dfrac{\dfrac{\Sigma_0}{B_0}\cdots\dfrac{\Sigma_n}{B_n}}{\{A\}_I}(Rule)\right) \overset{\text{def}}{=} \dfrac{Mark\left(\dfrac{\Sigma_0}{\{B_0\}_{J_0}}\right)\cdots Mark\left(\dfrac{\Sigma_n}{\{B_n\}_{J_n}}\right)}{\{A\}_I}(Rule)$$

The following are the finite natural number set operations used in *Mark*:

$I+n \overset{\text{def}}{=} \{x+n \mid x \in I\}$ $\quad$ $I-n \overset{\text{def}}{=} \{x-n \mid x-n \geq 0, x \in I\}$

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

- Definition of *Mark*

$$Mark\left(\dfrac{\dfrac{\Sigma_0}{B_0}\ \cdots\ \dfrac{\Sigma_n}{B_n}}{\{A\}_\phi}\right) \overset{\text{def}}{=} \dfrac{Mark\left(\dfrac{\Sigma_0}{\{B_0\}_\phi}\right)\ \cdots\ Mark\left(\dfrac{\Sigma_n}{\{B_n\}_\phi}\right)}{\{A\}_\phi}$$

Assume $I \neq \phi$ in the following.

$$Mark\left(\dfrac{\dfrac{\Sigma_0}{t}\quad\dfrac{\Sigma_1}{A(t)}}{\{\exists x.\ A(x)\}_I}(\exists\text{-}I)\right) \overset{\text{def}}{=} \dfrac{Mark\left(\dfrac{\Sigma_0}{\{t\}_K}\right)\quad Mark\left(\dfrac{\Sigma_1}{\{A(t)\}_{I-1}}\right)}{\{\exists x.\ A(x)\}_I}(\exists\text{-}I)$$

where $K \overset{\text{def}}{=} \begin{cases} \phi & \text{if } 0 \notin I; \\ \{0\} & \text{otherwise.} \end{cases}$

$$Mark\left(\dfrac{\dfrac{\Sigma_0}{\exists x.\ A(x)}\qquad \dfrac{\begin{array}{c}[x,\ A(x)]\\ \Sigma_1\end{array}}{C}}{\{C\}_I}(\exists\text{-}E)\right) \overset{\text{def}}{=} \dfrac{Mark\left(\dfrac{\Sigma_0}{\{\exists x.\ A(x)\}_K}\right)\quad Mark\left(\dfrac{\Sigma_1}{\{C\}_I}\right)}{\{C\}_I}(\exists\text{-}E)$$

where $K = \begin{cases} M+1 & \text{if } L = \phi \\ \{0\} \cup (M+1) & \text{if } L = \{0\} \end{cases}$

and L and M are the unions of the markings of all the occurrences of x and $A(x)$ as hypotheses obtained in $Mark(\Sigma_1/\{C\}_I)$.

$$Mark\left(\dfrac{\dfrac{\begin{array}{c}[x:\sigma]\\ \Sigma\end{array}}{A(x)}}{\{\forall x:\sigma.\ A(x)\}_I}(\forall\text{-}I)\right) \overset{\text{def}}{=} \dfrac{Mark\left(\dfrac{\Sigma}{\{A(x)\}_I}\right)}{\{\forall x:\sigma.\ A(x)\}_I}(\forall\text{-}I)$$

$$Mark\left(\dfrac{\dfrac{\Sigma_0}{t}\qquad\dfrac{\Sigma_1}{\forall x.\ A(x)}}{\{A(t)\}_I}(\forall\text{-}E)\right)\stackrel{\text{def}}{=}\dfrac{Mark\left(\dfrac{\Sigma_0}{\{t\}_{\{0\}}}\right)\quad Mark\left(\dfrac{\Sigma_1}{\{\forall x.\ A(x)\}_I}\right)}{\{A(t)\}_I}(\forall\text{-}E)$$

$$Mark\left(\dfrac{\dfrac{\Sigma_0}{A}\quad\dfrac{\Sigma_1}{B}}{\{A\land B\}_I}(\land\text{-}I)\right)\stackrel{\text{def}}{=}\dfrac{Mark\left(\dfrac{\Sigma_0}{\{A\}_{I(<l(A))}}\right)\quad Mark\left(\dfrac{\Sigma_1}{\{B\}_{I(\geq l(A))-l(A)}}\right)}{\{A\land B\}_I}(\land\text{-}I)$$

$$Mark\left(\dfrac{\dfrac{\Sigma}{A\land B}}{\{A\}_I}(\land\text{-}E)_0\right)\stackrel{\text{def}}{=}\dfrac{Mark\,(\Sigma/\{A\land B\}_I)}{\{A\}_I}(\land\text{-}E)_0$$

$$Mark\left(\dfrac{\dfrac{\Sigma}{A\land B}}{\{B\}_I}(\land\text{-}E)_1\right)\stackrel{\text{def}}{=}\dfrac{Mark\,(\Sigma/\{A\land B\}_{I+l(A)})}{\{B\}_I}(\land\text{-}E)_1$$

$$Mark\left(\dfrac{\dfrac{\Sigma}{A}}{\{A\lor B\}_I}(\lor\text{-}I)_0\right)\stackrel{\text{def}}{=}\dfrac{Mark\,(\Sigma/\{A\}_{(I-1)(<l(A))})}{\{A\lor B\}_I}(\lor\text{-}I)_0$$

$$Mark\left(\dfrac{\dfrac{\Sigma}{B}}{\{A\lor B\}_I}(\lor\text{-}I)_1\right)\stackrel{\text{def}}{=}\dfrac{Mark\,(\Sigma/\{B\}_{I-(l(A)+1)})}{\{A\lor B\}_I}(\lor\text{-}I)_1$$

$$Mark\left(\dfrac{\dfrac{\Sigma_0}{A\lor B}\quad\dfrac{\overset{[A]}{\Sigma_1}}{C}\quad\dfrac{\overset{[B]}{\Sigma_2}}{C}}{\{C\}_I}(\lor\text{-}E)\right)$$

$$\stackrel{\text{def}}{=}\dfrac{Mark\,(\Sigma_0/\{A\lor B\}_K)\quad Mark\,(\Sigma_1/\{C\}_I)\quad Mark\,(\Sigma_2/\{C\}_I)}{\{C\}_I}(\lor\text{-}E)$$

where $K=\{0\}\cup(J_0+1)\cup(J_1+1+l(A))$, and J_0 and J_1 are the unions of the markings of all the occurrences of A and B as hypotheses. Note that for the case of the modified $\lor$-code, both J_0 and J_1 are ϕ, so that $K=\{0\}$ if $I\neq\phi$

$$Mark\left(\dfrac{\dfrac{\overset{[A]}{\Sigma}}{B}}{\{A\supset B\}_I}(\supset\text{-}I)\right)\stackrel{\text{def}}{=}\dfrac{Mark\left(\dfrac{\Sigma}{\{B\}_I}\right)}{\{A\supset B\}_I}(\supset\text{-}I)$$

$$Mark\left(\dfrac{\dfrac{\Sigma_0}{A} \qquad \dfrac{\Sigma_1}{A \supset B}}{\{B\}_I}(\supset\text{-}E)\right) \overset{\text{def}}{=} \dfrac{Mark\,(\Sigma_0/\{A\}_{TRV}) \quad Mark\,(\Sigma_1/\{A \supset B\}_I)}{\{B\}_I}(\supset\text{-}E)$$

$$Mark\left(\dfrac{\dfrac{\Sigma_0}{A(0)} \qquad \dfrac{\begin{array}{c}[A(pred(x))]\\ \Sigma_1\end{array}}{A(x)}}{\{\forall x.\ A(x)\}_I}(nat\text{-}ind)\right)$$

$$\overset{\text{def}}{=} \dfrac{Mark\,(\Sigma_0/\,\{A(0)\}_I) \quad Mark\,(\Sigma_1/\{A(x)\}_I)}{\{\forall x.\ A(x)\}_I}(nat\text{-}ind)$$

$$Mark\left(\dfrac{\dfrac{\Sigma_0}{x = y} \quad \dfrac{\Sigma_1}{A(x)}}{\{A(y)\}_I}(=\text{-}E)\right) \overset{\text{def}}{=} \dfrac{Mark\left(\dfrac{\Sigma_0}{\{x = y\}_\phi}\right) \quad Mark\left(\dfrac{\Sigma}{\{A(x)\}_I}\right)}{\{A(y)\}_I}(=\text{-}E)$$

$$Mark\left(\dfrac{\dfrac{\Sigma}{\perp}}{\{A\}_I}(\perp\text{-}E)\right) \overset{\text{def}}{=} \dfrac{Mark\,(\Sigma/\{\perp\}_\phi)}{\{A\}_I}(\perp\text{-}E)$$

- Assumption

$Mark(\{A\}_I) \overset{\text{def}}{=} \{A\}_I$

- Inference on terms

$Mark\left(\dfrac{\Sigma}{\{t : \sigma\}_{\{0\}}}(*)\right) \overset{\text{def}}{=}$ Let the marking of the form $s : \tau$ in Σ which s occurs in t be $\{0\}$ and the marking of other nodes in Σ be ϕ

$Mark$ is well-defined, i.e., the set, I, attached to a node, A, by $Mark$ is a subset of $\{0, 1, \cdots, l(A) - 1\}$ or, if A is in the form of $M : \sigma$, $\{0\}$.

4. Marking Procedure on Induction Proofs

4.1 Marking Condition

The programs extracted from induction proofs are recursive call programs. Assume that the declaration, I, is given to an induction proof and that $Mark$ is performed with the declaration. Let J be the union of the markings of all the occurrences of induction hypothesis.

$$\dfrac{\dfrac{\Sigma_0}{\{A(0)\}_I} \qquad \dfrac{\begin{array}{c}\{[A(pred(x))]\}_J\\ \Sigma_1\end{array}}{\{A(x)\}_I}}{\{\forall x.\ A(x)\}_I}(nat\text{-}ind)$$

The recursive call program, f, extracted from the marked proof tree should calculate part of the realizer sequence of $A(x)$ (conclusion of the induction step) of the positions specified by I, if the input is not 0. At the recursive call step, it should calculate the realizer sequence of $A(pred(x))$ (induction hypothesis) of a set of positions which is included in I. In other words, J must be a subset of I, $J \subseteq I$. This condition will be called the *marking condition*. This raises a question: does the marking condition always hold? In fact, the answer is not always affirmative. The next subsection gives the way to overcome the situation in which marking condition does not hold, and proof theoretic characterization of the critical cases will also be given after the next subsection.

4.2 Marking with Backtracking

The basic idea to overcome the situation in which the marking condition does not hold is *marking procedure controlled by backtracking*: Let a marked induction proof tree be as in the previous subsection. If $J \not\subseteq I$, then enlarge I to $I \cup J$ and perform $Mark$ again. Then, it may happen that J is enlarged to J' and $J' \not\subseteq I \cup J$. In this case, $I \cup J$ must be enlarged again to $I \cup J \cup J' (= I \cup J')$. This procedure will be continued until the marking condition is satisfied, but the procedure always terminates because the declaration $I \cup J \cup J' \cup \cdots$ is bound by TRV.

The situation is a little complex for the nested induction. Assume that an induction proof Π_0 contains another induction proof Π_1 in it. Let I be the declaration to Π_0, and perform the marking procedure. Let J, LL, and L be the unions of the markings of all the occurrences of the induction the hypotheses of Π_0 and Π_1, and the marking of the conclusion of Π_1. The marking conditions for the nested induction are $J \subseteq I$ (condition for Π_0) and $LL \subseteq L$ (condition for Π_1). I must be made sufficiently large to satisfy both of the conditions. Generally speaking, J, L, and LL are enlarged when I is enlarged. Suppose, for example, that $LL \subseteq L$ and $J \not\subseteq I$. Then, I must be enlarged to satisfy the condition for Π_0. However, this procedure may destroy the condition for Π_1: $LL' \not\subseteq L'$ may hold for the new values, LL' and L', of LL and L. Then, I must be enlarged again to satisfy the condition for Π_1, and that may destroy the condition for Π_0, and so on. Therefore, backtracking becomes rather complicated for the nested induction.

However, if the induction hypothesis of Π_0 is not used in Π_1, the backtrack can be made simpler by using a sort of projection function:

1) Let the declaration, I, to Π_0 be sufficiently large to satisfy the marking condition for Π_0, and let L and LL be as above;

2) If $LL \subseteq L$, the marking procedure on Π_0 is successful. Otherwise, go to 3);

3) Enlarge L (not I) to L' to satisfy the marking condition for Π_1, which is to say $Mark(\Pi_1)$ succeeds.

The modified proof compilation algorithm will become a little complex if it is to handle the marked proof tree obtained by the procedure 1) and 3) . The proof compiler will generate the following program from the marked version of Π_0:

$$\mu(z_{i_0}, \cdots, z_{i_k}).\ F_T$$

where $\{i_0, \cdots, i_k\} = I$, and F is the term in which T, the code from the marked version of Π_1, occurs. T is obtained as follows: Let S be the realizer code extracted from the marked version of the subproof Π_1. S is the realizer codes of the conclusion of Π_1 of the positions specified by $L'(\supset L)$. Then, $T \overset{\text{def}}{=} proj(L/L')(S)$ which works as follows: First, evaluate a value of S. Let $(s_{i_1}, \cdots, s_{i_k})$ be the value, and $L' = \{i_1, \cdots, i_k\} \supset \{j_1, \cdots, j_l\} = L$. Then the value of $proj(L/L')(S)$ is $(s_{j_1}, \cdots, s_{j_l})$

4.3 Proof theoretic characterization of critical applications

This subsection gives a proof theoretic characterization of the situation in which the marking condition does not hold. The results have no direct relation with the proof compilation algorithm that generates redundancy-free programs. However, the characterization gives a proof theoretic explanation of the phenomenon of marking of proof trees. Also, it could give a way of program analysis of recursive call programs at proof level.

4.3.1 Critical segments

(1) An example

Let $A(x) \overset{\text{def}}{=} \exists y.B(x, y) \vee C(x, y)$. Suppose that $\forall x : nat.A(x)$ is proved by mathematical induction, and the induction step proceeds as follows:

$$\dfrac{[\exists y.B(x-1, y) \vee C(x-1, y)] \quad \Pi}{A(x)}(\exists\text{-}E)$$

where $\exists y.B(x-1, y) \vee C(x-1, y)$ is the induction hypothesis, and Π is as follows:

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

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

$$\dfrac{\{[\exists y.\ B(x-1, y) \vee C(x-1, y)]\}_L \qquad \Pi'}{\{A(x)\}_{\{0\}}}(\exists\text{-}E)$$

where Π' is as follows:

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

where Σ_{00} and Σ_{11} are the marked versions of Σ_0 and Σ_1. By the definition of *Mark*, K contains 0, and then, L contains 1. Therefore, the marking condition does not hold: $L \not\subseteq \{0\}$. This indicates that the marking condition does not always hold when $(\vee\text{-}E)$ and $(\exists\text{-}E)$ are used below the deduction sequence down from the induction hypotheses.

(2) Formal definition of critical segments

The reason for this phenomenon is that the realizer code of $A \vee B$ consists not only of the code of A and B but also of the code, *left* or *right*. Therefore, the marking of $A \vee B$ must contain 0 when the formula is the major premise of a $(\vee\text{-}E)$ application.

The following proof theoretic terminologies are needed to formalize critical segments.

Definition 16: *Thread*

A *thread* is a consecutive formula occurrences in a proof tree from a top-formula to the end-formula.

Definition 17: *Segment*

The same formulas occur as minor premises and conclusions in $(\vee\text{-}E)$ and $(\exists\text{-}E)$ rules. Therefore, if there are successive applications of these rules in a proof tree, there are consecutive occurrences of the same formula in a thread. This sequence is called a *segment*. Any formula occurrence in a proof tree which is not a minor premise or a conclusion of these rules is also regarded as forming a *trivial* segment.

Definition 18: *Path*

A *path* is the deduction sequence from a top-formula which is not discharged by $(\vee\text{-}E)$ or $(\exists\text{-}E)$ applications to the end-formula or to a minor premise of an application of the $(\supset\text{-}E)$ rule. A path branches at an application of the $(\vee\text{-}E)$ rule or the $(\exists\text{-}E)$ rule:

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

In the ($\vee$-E) rule application above, a path from a top-formula in Σ_0 branches at $A \vee B$. A branch passes through an occurrence of A or B as the discharged hypotheses, and goes down to the occurrence of C as the conclusion of the application. It is similar in the ($\exists$-E) rule application: A path from a top-formula in Σ_0' branches at $\exists x.\ A(x)$, and a branch goes to the occurrence of C through one of the occurrences of $A(x)$ as the discharged hypotheses. A path whose last element is the end-formula of the proof tree is called a *main path*.

See [Prawitz 65] for the formal definitions of thread, segment and path.

Definition 19: *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 called the *major premise attached to* A.

Definition 20: *Proper segment*
The non-trivial segment in a marked proof tree, Π, is called *proper* if every formula occurrence in the segment has non-nil marking.

Definition 21: *Critical segments:*
Let Π be an induction step proof in a proof tree. A proper segment, σ, in Π is *critical* if 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 in Π from an occurrence of the induction hypothesis which also passes through σ.

Definition 22: *Indispensable marking numbers:*
Assume an induction step proof, Π. An *indispensable marking number* is a marking number of a node in Π which is obtained as follows:
a) The node is along a main path in Π from an occurrence of the induction hypothesis;
b) The marking number is propagated from the marking number, 0, of an occurrence of a $\vee$ formula as the major premise of an ($\vee$-E) application.

If there is a critical segment in an induction proof, there is a possibility that the marking condition is not satisfied because of the indispensable marking numbers of occurrences of the induction hypothesis.
Indispensable marking numbers can be calculated systematically in a restricted case as in the following lemma.

Lemma 2:
Let Π be an induction step proof. Let $S \overset{\text{def}}{=} (A_1, A_2, \cdots, A_m)$ be a critical segment in Π, and π be a main path in Π from an occurrence of the induction hypothesis which passes through S and a major premise, F, attached to A_n (for some n, $1 \le n \le m$) in S. Assume that there is a subsequence, $\pi_0 = \{B_1, B_2, \cdots, B_l\}$, of π such that:
a) $B_1 = F$

b) B_i is a major premise attached to $A_{k(i)}$ in S ($1 \leq i \leq l$, $k(1) = n$)

c) B_i ($i \geq 2$) is discharged by the ($\exists$-E) or ($\vee$-E) application whose minor premise (or one of whose minor premises) is $A_{k(i-1)}$.

Then, the marking of F contains the marking numbers $\psi(j)$ ($1 \leq j \leq K$) defined as follows:

$$\varphi(i) \stackrel{\text{def}}{=} \sum_{p=1}^{\alpha(i)} \psi(p)$$

$$\psi(p) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p = 1 \\ l(C) + 1 & \text{if } B_{p-1} (= C \vee D) \text{ is a major premise of } (\vee\text{-}E) \text{ and } B_p = D \\ 1 & \text{otherwise} \end{cases}$$

where K and $\alpha(i)$ are as follows: Let $\pi_1 = \{B_{\alpha(1)}, \cdots, B_{\alpha(K)}\}$ is the subsequence of π_0 such that $B_{\alpha(j)}$ ($1 \leq j \leq K$) is a major premise of an application of the ($\vee$-E) rule.

Proof: Let the occurrences of A_i and A_{i+1} be as follows:

$$\cfrac{A \vee B \quad \cfrac{\begin{matrix}[A]\\ \Sigma_1\end{matrix}}{A_i} \quad \cfrac{\begin{matrix}[B]\\ \Sigma_2\end{matrix}}{A_i'}}{\begin{matrix}A_{i+1}\\ \Pi_1\end{matrix}}\,(\vee\text{-}E) \qquad (\text{where } A_i = A_i' = A_{i+1})$$

Assume that $A \vee B$ is an element of π_1. As S is a proper segment, the marking of $A \vee B$ contains 0.

Case 1: Assume that an element, A_k ($k \geq i+1$), in S is a minor premise of an application of the ($\exists$-E) rule and F_0 in π_0 is a major premise attached to A_k. Assume also that F_0 is immediately before $A \vee B$ in π, that is, $A \vee B$ is discharged by the application of ($\exists$-E). Then, the marking number, 0, of $A \vee B$ becomes 1 in the marking of F_0:

$$\cfrac{\{F_0\}_{\{1\}\cup T_1} \qquad \begin{matrix}\cfrac{\{[A \vee B]\}_{\{0\}\cup T_0} \quad \cfrac{\begin{matrix}[A]\\ \Sigma_1\end{matrix}}{\{A_i\}_I} \quad \cfrac{\begin{matrix}[B]\\ \Sigma_2\end{matrix}}{\{A_i'\}_I}}{\{A_{i+1}\}_I}\\ \cdots \\ \{A_k\}_I\end{matrix}}{\{A_{k+1}\}_I}\,(\exists\text{-}E)$$

Case 2: Assume that an element, A_k ($k \geq i+1$), in S is a left minor premise of an ($\vee$-E) application and F_1 in π_0 is a major premise attached to A_k. Assume also that F_1 is immediately before $A \vee B$ in π. Then, the marking number, 0, of $A \vee B$ become 1 in the marking of F_1.

$$\cfrac{\{F_1\}_{\{1\}\cup T_1} \qquad \begin{matrix}\cfrac{\{[A \vee B]\}_{\{0\}\cup T_0} \quad \cfrac{\begin{matrix}[A]\\ \Sigma_1\end{matrix}}{\{A_i\}_I} \quad \cfrac{\begin{matrix}[B]\\ \Sigma_2\end{matrix}}{\{A_i'\}_I}}{\{A_{i+1}\}_I}\\ \cdots \\ \{A_k\}_I\end{matrix} \qquad \Pi'}{\{A_{k+1}\}_I}\,(\vee\text{-}E)$$

Case 3: Assume that an element, A_k ($k \geq i+1$), in S is a right minor premise of a ($\vee$-E) application and $F_2(= C \vee D)$ in π_0 is a major premise attached to A_k. Assume also that F_2 is immediately before $A \vee B$ in π. Then, the marking number, 0, of $A \vee B$ becomes $l(C)+1$ in the marking of F_2.

$$\dfrac{\{F_2\}_{\{l(C)+1\}\cup T_1} \qquad \Pi' \qquad \begin{array}{c} \dfrac{\{[A \vee B]\}_{\{0\}\cup T_0} \quad \dfrac{\begin{array}{c}[A]\\ \Sigma_1\end{array}}{\{A_i\}_I} \quad \dfrac{\begin{array}{c}[B]\\ \Sigma_2\end{array}}{\{A_i'\}_I}}{\{A_{i+1}\}_I} \\ \cdots \\ \{A_k\}_I \end{array}}{\{A_{k+1}\}_I}(\vee\text{-}E)$$

The lemma follows by continuing the discussion in a similar way. ▮

Example 5: There are many other cases of indispensable marking numbers. Assume the following induction step proof where $F = \exists x.A \wedge ((B \vee C) \vee D)$, IH is an occurrence of the induction hypothesis and $I \neq \phi$.

$$\begin{array}{c}\dfrac{\dfrac{\begin{array}{c}[IH]\\ \Sigma_0\end{array}}{\{F\}_N} \quad \dfrac{\dfrac{\{[A \wedge ((B \vee C) \vee D)]\}_M}{\{(B \vee C) \vee D\}_L}(\wedge\text{-}E) \quad \dfrac{\{[B \vee C]\}_K \quad \dfrac{\begin{array}{c}[B]\\ \Sigma_1\end{array}}{\{A\}_I} \quad \dfrac{\begin{array}{c}[C]\\ \Sigma_2\end{array}}{\{A\}_I}}{\{A\}_I}(\vee\text{-}E) \quad \dfrac{\begin{array}{c}[D]\\ \Sigma_3\end{array}}{\{A\}_I}}{\{A\}_I}(\vee\text{-}E)}{\{A\}_I}(\exists\text{-}E)\\ \Pi\end{array}$$

As $0 \in K$, $1 \in L$ and $l(A)+1 \in M$. Hence, $l(A)+2 \in N$. 0, 1, $l(A)+1$, and $l(A)+2$ are indispensable marking numbers of $B \vee C$, $(B \vee C) \vee D$, $A \wedge ((B \vee C) \vee D)$, and F.

4.3.2 Critical ($\exists$-E) application

(1) An example
Assume that $\forall x.\exists y.\exists z.\ A(x,y,z)$ is proved in mathematical induction, and the declaration, $\{0\}$ is given to the conclusion. Also assume that the induction step part of the marked proof tree is as follows:

$$\dfrac{\{[\exists y.\exists z.A(x-1,y,z)]\}_L \quad \Pi}{\{\exists y.\exists z.A(x,y,z)\}_{\{0\}}}(\exists\text{-}E)$$

where Π is as follows:

$$\dfrac{\{[\exists z.A(x-1,y,z)]\}_K \quad \dfrac{\dfrac{\begin{array}{c}\{[y]\}_{\{0\}}\\ \{[z]\}_{\{0\}}\\ \Sigma_0\end{array}}{\{s_{y,z}\}_{\{0\}}} \quad \dfrac{\begin{array}{c}\{[y]\}_\phi\{[z]\}_\phi\\ \{[A(x-1,y,z)]\}_\phi\\ \Sigma_1\end{array}}{\{\exists z.A(x,s_{y,z},z)\}_\phi}}{\{\exists y.\exists z.A(x,y,z)\}_{\{0\}}}(\exists\text{-}I)}{\{\exists y.\exists z.A(x,y,z)\}_{\{0\}}}(\exists\text{-}E)$$

Note that both of the assumptions of the ($\exists$-E) rules (eigenvariables), y and z, are used to construct $s_{y,z}$, so that $0 \in K$ and $\{0,1\} \subseteq L$. Therefore, the marking condition does not hold: $L \not\subseteq \{0\}$.

(2) Definition of critical ($\exists$-E) applications

Definition 23: *Critical* ($\exists$-E) *applications*
If a $\forall$ formula $A \overset{\text{def}}{=} \forall x.\ B(x)$ is proved in induction and A contains an $\exists$ formula $C(x)$. Assume that there is a main path from an occurrence of the induction hypothesis in which $C(x-1)$ occurs as the major premise of an ($\exists$-E) application and that there is an eigenvariable of the application whose marking is $\{0\}$. Let k be the position number of the principal sign, $\exists$, of $C(x)$ in A. Then, if k is not contained in the declaration to A, the ($\exists$-E) application is said to be critical.

Note that, in the example (1), one of the ($\exists$-E) applications is critical.

4.3.3 Other critical applications

The notion of critical segments and critical ($\exists$-E) applications can only capture the situation of the marking along a main path from an occurrence of the induction hypothesis. However, there may be a path from an occurrence of the induction hypothesis which is not a main path. For example, assume a marked induction step proof is as follows:

$$\dfrac{\dfrac{\begin{array}{c}\{[A(x-1)]\}_K\\ \Sigma_0\end{array}}{\{B\}_{TRV}} \qquad \dfrac{\Sigma_1}{\{B \supset C\}_J}}{\begin{array}{c}\{C\}_J\\ \Pi\\ \{A(x)\}_I\end{array}}(\supset\text{-}E) \qquad (J \neq \phi)$$

The marking of B as a minor premise of the application of ($\supset$-E) is always TRV, so that K is always the same value whenever J is not nil marking. Therefore, I must be made sufficiently large to satisfy, $K \subseteq I$.

5. Modified Proof Compilation Algorithm

The proof compilation should be modified to handle marked proof trees. The chief modification is:
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 the formula, A, is marked by ϕ, no code should be extracted and there is no need to analyze the subtree determined by A.
The following is the definition of the modified version of the Ext procedure, $NExt$. $|I|$ denotes the number of elements in I.

(1) Nil marking:

$$NExt\left(\frac{\{A_0\}_{J_0}\cdots\{A_k\}_{J_k}}{\{B\}_I}(Rule)\right) \overset{\text{def}}{=} () \quad \text{where } I=\phi$$

In the following, I is assumed to be non-nil.

(2) Assumptions:

$$NExt(\{A\}_I) \overset{\text{def}}{=} proj(I)(Rv(A))$$

(3) $\wedge$ and $\vee$ formulas:

- $$NExt\left(\frac{\dfrac{\Sigma_0}{\{A_0\}_{I_0}} \quad \dfrac{\Sigma_1}{\{A_1\}_{I_1}}}{\{A_0\wedge A_1\}_I}(\wedge\text{-}I)\right) \overset{\text{def}}{=} \left(NExt\left(\frac{\Sigma_0}{\{A_0\}_{I_0}}\right), NExt\left(\frac{\Sigma_1}{\{A_1\}_{I_1}}\right)\right)$$

- $$NExt\left(\frac{\dfrac{\Sigma}{\{A_0\wedge A_1\}_J}}{\{A_i\}_I}(\wedge\text{-}E)_i\right) \overset{\text{def}}{=} NExt\left(\frac{\Sigma}{\{A_0\wedge A_1\}_J}\right) \qquad \text{where } i=0,1.$$

- $$NExt\left(\frac{\dfrac{\Sigma}{\{A\}_J}}{\{A\vee B\}_I}(\vee\text{-}I)_0\right) \overset{\text{def}}{=} \begin{cases} \left(left, NExt\left(\dfrac{\Sigma}{\{A\}_J}\right), any[k]\right) & \text{if } 0\in I \\ \left(NExt\left(\dfrac{\Sigma}{\{A\}_J}\right), any[l]\right) & \text{if } 0\notin I \end{cases}$$

- $$NExt\left(\frac{\dfrac{\Sigma}{\{B\}_J}}{\{A\vee B\}_I}(\vee\text{-}I)_1\right) \overset{\text{def}}{=} \begin{cases} (right, any[k], NExt\left(\dfrac{\Sigma}{\{B\}_J}\right)) & \text{if } 0\in I \\ (any[l], NExt\left(\dfrac{\Sigma}{\{B\}_J}\right)) & \text{if } 0\notin I \end{cases}$$

where $k=\mid I\mid -(1+\mid J\mid)$ and $l=\mid I\mid - \mid J\mid$.

(4) The code from ($\vee$-E) rule:

$$NExt\left(\frac{\dfrac{\Sigma_0}{\{A\vee B\}_{J_0}} \quad \dfrac{\begin{array}{c}\{[A]\}_{J_1}\\ \Sigma_1\end{array}}{\{C\}_I} \quad \dfrac{\begin{array}{c}\{[B]\}_{J_2}\\ \Sigma_2\end{array}}{\{C\}_I}}{\{C\}_I}(\vee\text{-}E)\right)$$

is as follows:

a) $if\ beval(A)\ then\ NExt\left(\dfrac{\Sigma_1}{\{C\}_I}\right)\ else\ NExt\left(\dfrac{\Sigma_2}{\{C\}_I}\right)$ [**modified $\vee$ code**]

when both A and B are equations or inequations of terms

Note that, in this case, $J_1=J_2=\phi$.

b) $if\ left=proj(0)\left(NExt\left(\dfrac{\Sigma_0}{\{A\vee B\}_{J_0}}\right)\right)\ then\ NExt\left(\dfrac{\Sigma_1}{\{C\}_I}\right)\theta\ else\ NExt\left(\dfrac{\Sigma_2}{\{C\}_I}\right)\theta$

otherwise

where $\theta \stackrel{\text{def}}{=} \left\{ \begin{array}{l} proj(J_1)(Rv(A))/ttseq(1,|J_1|)\,(NExt\,(\Sigma_0/\{A \vee B\}_{J_0}))\,, \\ proj(J_2)(Rv(B))/tseq(|J_1|+1)\,(NExt\,(\Sigma_0/\{A \vee B\}_{J_0})) \end{array} \right\}$

(5) The codes from the ($\supset$-I) and ($\forall$-I) rules:

$$\bullet\ NExt \left(\dfrac{\begin{array}{c}[x:\sigma] \\ \dfrac{\Sigma}{\{A(x)\}_I}\end{array}}{\{\forall x:\sigma.\ A(x)\}_I}(\forall\text{-}I) \right) \stackrel{\text{def}}{=} \lambda x.\ NExt\left(\dfrac{\Sigma}{\{A(x)\}_I}\right)$$

$$\bullet\ NExt \left(\dfrac{\begin{array}{c}\{[A]\}_J \\ \dfrac{\Sigma}{\{B\}_I}\end{array}}{\{A \supset B\}_I}(\supset\text{-}I) \right) \stackrel{\text{def}}{=} \lambda\ Rv(A).\ NExt\left(\dfrac{\Sigma}{\{B\}_I}\right)$$

(6) The codes from the proofs in ($\supset$-E) and ($\forall$-E):

$$\bullet\ NExt \left(\dfrac{\dfrac{\Sigma_0}{\{A\}_{TRV}} \quad \dfrac{\Sigma_1}{\{A \supset B\}_I}}{\{B\}_I}(\supset\text{-}E) \right) \stackrel{\text{def}}{=} NExt\left(\dfrac{\Sigma_1}{\{A \supset B\}_I}\right)\left(NExt\left(\dfrac{\Sigma_0}{\{A\}_{TRV}}\right)\right)$$

$$\bullet\ NExt \left(\dfrac{\dfrac{\Sigma_0}{\{t:\sigma\}_{\{0\}}} \quad \dfrac{\Sigma_1}{\{\forall x:\sigma.\ A(x)\}_I}}{\{A(t)\}_I}(\forall\text{-}E) \right) \stackrel{\text{def}}{=} NExt\left(\dfrac{\Sigma_1}{\{\forall x:\sigma.\ A(x)\}_I}\right)(t)$$

(7) The codes from the ($\exists$-I) and ($\exists$-E) rules:

$$\bullet\ NExt \left(\dfrac{\dfrac{\Sigma_0}{\{t:\sigma\}_J} \quad \dfrac{\Sigma_1}{\{A(t)\}_K}}{\{\exists x:\sigma.\ A(x)\}_I}(\exists\text{-}I) \right) \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} \left(t, NExt\left(\dfrac{\Sigma_1}{\{A(t)\}_K}\right)\right) & \text{if } J = \{0\} \\ NExt\left(\dfrac{\Sigma_1}{\{A(t)\}_K}\right) & \text{if } J = \phi \end{array} \right.$$

$$\bullet\ NExt \left(\dfrac{\dfrac{\Sigma_0}{\{\exists x:\sigma.\ A(x)\}_J} \quad \begin{array}{c}\{[x:\sigma]\}_K, \{[A(x)]\}_L \\ \dfrac{\Sigma_1}{\{C\}_I}\end{array}}{\{C\}_I}(\exists\text{-}E) \right) \stackrel{\text{def}}{=} NExt\left(\dfrac{\Sigma_1}{\{C\}_I}\right)\theta$$

where $\theta \stackrel{\text{def}}{=} \left\{ \begin{array}{c} proj(L)(Rv(A(x)))/tseq(1)\,(NExt\,(\Sigma_0/\{\exists x:\sigma.\ A(x)\}_J))\,, \\ x/proj(0)\,(NExt\,(\Sigma_0/\{\exists x:\sigma.\ A(x)\}_J)) \end{array} \right\}$ if $0 \in J$, and $\theta \overset{\text{def}}{=} \{proj(L)(Rv(A(x)))/NExt(\Sigma_0/\{\exists x : \sigma.\ A(x)\}_J)\}$ if $0 \notin J$.

(8) The code extracted from a proof in ($\perp$-E) rule:

$$\bullet\ NExt\left(\dfrac{\dfrac{\Sigma}{\perp}}{\{A\}_I}(\perp\text{-}E)\right) \overset{\text{def}}{=} any[k] \qquad\qquad \text{where } k = |I|$$

(9) The code extracted from (= -E) rule:

$$\bullet\ NExt\left(\dfrac{\dfrac{\Sigma_0}{\{x = y\}_\phi} \quad \dfrac{\Sigma_1}{\{A(x)\}_I}}{\{A(y)\}_I}(= \text{-}E)\right) \overset{\text{def}}{=} NExt\left(\dfrac{\Sigma_1}{\{A(x)\}_I}\right)$$

(10) The realizer code extracted from the proof by mathematical induction:

$$\bullet\ NExt\left(\dfrac{\dfrac{\Sigma_0}{\{A(0)\}_I} \qquad \begin{array}{c}\{[x : nat]\}_K\{[x > 0]\}_\phi\{[A(x-1)]\}_J \\ \dfrac{\Sigma_1}{\{A(x)\}_I}\end{array}}{\{\forall x.\ A(x)\}_I}(nat\text{-}ind)\right)$$

$\overset{\text{def}}{=} \mu\, \overline{z}.\ \lambda\, x.\ if\ x = 0\ then\ NExt(\Sigma_0/\{A(0)\}_I)\ \ else\ NExt(\Sigma_1/\{A(x)\}_I)\sigma$
where $J \subseteq I$ and $\overline{z}$ is a sequence of fresh variables of length $|I|$ and
$\sigma \overset{\text{def}}{=} \{proj(I)(Rv(A(x-1)))/\overline{z}(pred(x))\}$

6. Some properties of $Mark$ and $NExt$

6.1 Normalization of marked proof trees

Let R be one of the logical connectives and quantifiers: $\supset$, $\wedge$, $\vee$, $\forall$, and $\exists$. An application of (R-I) succeeded by an application of (R-E) is called an R-*cut*. Cuts can be eliminated by the R-reduction rules as defined in [Prawitz 65] and the rules are used in proof normalization. The rules will be denoted red_R in the following.

Cuts can also be defined on marked proof trees: a part of a marked proof tree is called an R-cut if it is an R-cut when all the markings of the nodes are removed. The R-reduction rules on marked proof trees, which will be referred to as Red_R in the following, are defined as follows:

Definition 24: *R-reduction rules on marked proof trees*

- $Red_{\supset}$:

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\{[A]\}_J\\ \Sigma_0\end{matrix}}{\{B\}_I}}{\{A\supset B\}_I}(\supset\text{-}I)\qquad \dfrac{\Sigma_1}{\{A\}_{TRV}}}{\{B\}_I}(\supset\text{-}E)\quad\Longrightarrow\quad \dfrac{\begin{matrix}\dfrac{\Sigma_{11}}{\{[A]\}_J}\\ \Sigma_0\end{matrix}}{\{B\}_I}$$

- $Red_{\wedge}$:

$$\dfrac{\dfrac{\dfrac{\Sigma_0}{\{A\}_I}\quad\dfrac{\Sigma_1}{\{B\}_\phi}}{\{A\wedge B\}_I}(\wedge\text{-}I)}{\{A\}_I}(\wedge\text{-}E)_0\quad\Longrightarrow\quad\dfrac{\Sigma_0}{\{A\}_I}\qquad\qquad \dfrac{\dfrac{\dfrac{\Sigma_0}{\{A\}_\phi}\quad\dfrac{\Sigma_1}{\{B\}_I}}{\{A\wedge B\}_{I+l(A)}}(\wedge\text{-}I)}{\{B\}_I}(\wedge\text{-}E)_1\quad\Longrightarrow\quad\dfrac{\Sigma_1}{\{B\}_I}$$

- $Red_{\vee}$:

$$\dfrac{\dfrac{\dfrac{\Sigma_0}{\{A\}_{(K-1)(<l(A))}}}{\{A\vee B\}_K}(\vee\text{-}E)\qquad\dfrac{\begin{matrix}\{[A]\}_{J_0}\\ \Sigma_1\end{matrix}}{\{C\}_I}\qquad\dfrac{\begin{matrix}\{[B]\}_{J_1}\\ \Sigma_2\end{matrix}}{\{C\}_I}}{\{C\}_I}(\vee\text{-}E)\quad\Longrightarrow\quad\dfrac{\begin{matrix}\dfrac{\Sigma_{00}}{\{[A]\}_{J_0}}\\ \Sigma_1\end{matrix}}{\{C\}_I}$$

$$\dfrac{\dfrac{\dfrac{\Sigma_0}{\{B\}_{K-(l(A)+1)}}}{\{A\vee B\}_K}\qquad\dfrac{\begin{matrix}\{[A]\}_{J_0}\\ \Sigma_1\end{matrix}}{\{C\}_I}\qquad\dfrac{\begin{matrix}\{[B]\}_{J_1}\\ \Sigma_2\end{matrix}}{\{C\}_I}}{C}(\vee\text{-}E)\quad\Longrightarrow\quad\dfrac{\begin{matrix}\dfrac{\Sigma_{00}}{\{[B]\}_{J_1}}\\ \Sigma_2\end{matrix}}{\{C\}_I}$$

- $Red_{\forall}$:

$$\dfrac{\dfrac{\Sigma_0}{\{t\}_{\{0\}}}\qquad\dfrac{\dfrac{\begin{matrix}\{[x]\}_K\\ \Sigma_1\end{matrix}}{\{A(x)\}_I}}{\{\forall x.A(x)\}_I}(\forall\text{-}I)}{\{A(t)\}_I}(\forall\text{-}E)\quad\Longrightarrow\quad\dfrac{\begin{matrix}\dfrac{\Sigma_{00}}{\{[t]\}_K}\\ \Sigma_1\{x/t\}\end{matrix}}{\{A(t)\}_I}$$

- $Red_{\exists}$:

$$\dfrac{\dfrac{\dfrac{\Sigma_0}{\{t\}_M}\quad\dfrac{\Sigma_1}{\{A(t)\}_N}}{\{\exists x.A(x)\}_{LL}}(\exists\text{-}I)\qquad\dfrac{\begin{matrix}\{[x]\}_K\{[A(x)]\}_L\\ \Sigma_2\end{matrix}}{\{C\}_I}}{\{C\}_I}(\exists\text{-}E)\quad\Longrightarrow\quad\dfrac{\begin{matrix}\dfrac{\Sigma_{00}}{\{[t]\}_K}\;\dfrac{\Sigma_{11}}{\{[A(t)]\}_L}\\ \Sigma_2\{x/t\}\end{matrix}}{\{C\}_I}$$

The meaning of $(\Sigma_{11}/\{[A]\}_J/\Sigma_0/\{B\}_I)$ in the definition of $Red_{\supset}$ is as follows: Let $A_1,\cdots,A_k$ be the occurrences of A as hypothesis in $(\Sigma_0/\{B\}_I)$ and $J_1,\cdots,J_k$ be the marking of them such that $J_1\cup\cdots\cup J_k=J$. Then, $(\Sigma_{11}/\{[A]\}_J/\Sigma_0/\{B\}_I)$ is obtained by replacing the node A_i in $(\Sigma_0/\{B\}_I)$ by $Mark(\Sigma_1/\{A\}_{J_i})$ $(1 \leq i \leq k)$. The meaning of the tree obtained by $Red_\vee$, $Red_\forall$, and $Red_\exists$ are defined similarly.

R-reduction of proof trees and the marking procedure commute in the following sense:

Theorem 2: Normalization and $Mark$
Let Π be a proof and I be a declaration to Π, and let $Mark(I,\Pi)$ be the tree obtained by the marking procedure applied on Π with the declaration I. Assume that the last two applications of rules in Π form an R-cut. Then, if both $Mark(I,\Pi)$ and $Mark(I, red_R(\Pi))$ succeed, $Red_R(Mark(I,\Pi)) = Mark(I, red_R(\Pi))$, where R is $\supset$, $\wedge$, $\vee$, $\forall$, or $\exists$.

Proof: Straightforward∎

Note that $Mark$ fails when the proof contains induction proofs and the marking condition is not satisfied. Also, even if the marking of a normalized proof satisfies the marking condition, the condition is not always satisfied for the original proof. For example, assume the following is a marked version of an induction step proof, Π, with marking I:

$$\dfrac{\{[\exists y.B(x-1,y)]\}_K \qquad \dfrac{\dfrac{\overset{\{[y]\}_{\{0\}}}{\Sigma_0}}{\{t_y\}_{\{0\}}} \quad \dfrac{\dfrac{\overset{\{[z]\}_\phi}{\Sigma_1}}{\{A(z)\}_J}}{\{\forall z.A(z)\}_J}(\forall\text{-}I)}{\dfrac{\{A(t_y)\}_J}{\Pi_0}}(\forall\text{-}E)}{\{\exists y.\ B(x,y)\}_I}(\exists\text{-}E)$$

where $J \neq \phi$ and $0 \notin I$. Then, $0 \in K$, so that the application of $(\exists\text{-}E)$ is critical and the marking condition is not satisfied. However, if $red_\forall$ is applied to Π and the marking procedure is performed with the marking I, the tree is as follows:

$$\dfrac{\{[\exists y.\ B(x-1,y)]\}_{K'} \qquad \dfrac{\dfrac{\dfrac{\overset{\{[y]\}_\phi}{\Sigma_{00}}}{\overset{\{[t_y]\}_\phi}{\Sigma_1\{x/t_y\}}}}{\{A(t_y)\}_J}}{\Pi_0}}{\{\exists y.\ B(x,y)\}_I}(\exists\text{-}E)$$

In this case, $0 \notin K'$, so that the marking condition may be satisfied.

6.2 $NExt$ procedure and projection

Lemma 3: *Let the marked proof trees of $(\supset\text{-}I)$, $(\vee\text{-}E)$, and $(\exists\text{-}E)$ applications and an induction step proof be as follows:*

Π_0 $\qquad\qquad\qquad\qquad$ Π_1

$$\dfrac{\begin{array}{c}\{[A]\}_J\\ \underline{\quad\Sigma\quad}\\ \{B\}_I\end{array}}{\{A\supset B\}_I}(\supset\text{-}I) \qquad\qquad \dfrac{\begin{array}{c}\\ \underline{\quad\Sigma_0\quad}\\ \{A\vee B\}_K\end{array}\quad\begin{array}{c}\{[A]\}_{J_1}\\ \underline{\quad\Sigma_1\quad}\\ \{C\}_I\end{array}\quad\begin{array}{c}\{[B]\}_{J_2}\\ \underline{\quad\Sigma_2\quad}\\ \{C\}_I\end{array}}{\{C\}_I}(\vee\text{-}E)$$

Π_2 $\qquad\qquad\qquad\qquad\qquad\qquad$ Π_3

$$\dfrac{\begin{array}{c}\\ \underline{\quad\Sigma_0\quad}\\ \{\exists x.A(x)\}_K\end{array}\quad\begin{array}{c}\{[x]\}_L\{[A(x)]\}_M\\ \underline{\quad\Sigma_1\quad}\\ \{C\}_I\end{array}}{\{C\}_I}(\exists\text{-}E) \qquad\qquad \dfrac{\begin{array}{c}\\ \underline{\quad\Sigma_0\quad}\\ \{A(0)\}_I\end{array}\quad\begin{array}{c}\{[A(x-1)]\}_J\\ \underline{\quad\Sigma_1\quad}\\ \{A(x)\}_I\end{array}}{\{\forall x.A(x)\}_I}$$

where $\{[A]\}_J$ in Π_0, for example, means that J is the union of the marking of all the occurrences of A as the discharged hypothesis. Then,
(1) For Π_0, if an element, z, of $Rv(A)$ occurs free in $NExt(\Sigma/\{B\}_I)$, then z is an element of $proj(J)(Rv(A))$;
(2) For Π_1, if an element, z, of $Rv(A)$ (or $Rv(B)$) occurs free in $NExt(\Sigma_1/\{C\}_I)$ (or $NExt(\Sigma_2/\{C\}_I)$), then z is an element of $proj(J_1)(Rv(A))$ (or $proj(J_2)(Rv(B))$);
(3) For Π_2, if an element, z, of $Rv(A(x))$ occurs free in $NExt(\Sigma_1/\{C\}_I)$, then z is an element of $proj(M)(Rv(A(x)))$;
(4) For Π_3, if an element, z, of $Rv(A(x-1))$ occurs free in $NExt(\Sigma_1/\{A(x)\}_I)$, then z is an element of $proj(J)(Rv(A(x-1)))$.

Sketch of the proof: It is sufficient to prove the following somewhat stronger statement:

Let a marked proof tree, Π, be as follows, and let A be an arbitrary formula which is used in Π as a hypothesis and which is not discharged at any application of the rule in Π.

$$\dfrac{\begin{array}{c}\{A\}_{J_1}\\ \underline{\quad\Sigma_1\quad}\\ \{A\}_{L_1}\end{array}\ \cdots\ \begin{array}{c}\{A\}_{J_k}\\ \underline{\quad\Sigma_k\quad}\\ \{A\}_{L_k}\end{array}}{\{B\}_I}(R)$$

J_i is the union of the markings of all the occurrences of A as a hypothesis in Σ_i. Let $J \stackrel{\text{def}}{=} J_1 \cup \cdots \cup J_k$. Then,
a) all the variables in $proj(J)(Rv(A))$ occur free in $NExt(\Pi)$;
b) if $z \in Rv(A)$ and z occurs free in $NExt(\Pi)$, then z is an element of $proj(J)(Rv(A))$.

The proof is continued in induction on the construction of Π. If R is ($\wedge$-I), ($\wedge$-E), ($\vee$-I), ($\supset$-I), ($\supset$-E), ($\forall$-I), ($\forall$-E), or ($\exists$-I), the proof is straightforward from the definition of $NExt$. Assume that R is ($\vee$-E) and that Π is as follows:

$$\dfrac{\begin{array}{c}\\ \underline{\quad\Sigma_0\quad}\\ \{P\vee Q\}_K\end{array}\quad\begin{array}{c}\{[P]\}_{J_0}\\ \underline{\quad\Sigma_1\quad}\\ \{C\}_I\end{array}\quad\begin{array}{c}\{[Q]\}_{J_1}\\ \underline{\quad\Sigma_2\quad}\\ \{C\}_I\end{array}}{\{C\}_I}(\vee\text{-}E)$$

By the induction hypothesis, $NExt(\Sigma_1/\{C\}_I)$ (or $NExt(\Sigma_2/\{C\}_I)$) contains all the variables in $proj(J_0)(Rv(P))$ (or $proj(J_1)(Rv(Q))$). Therefore, by the definition of $NExt$, the whole of $tseq(1)(NExt(\Sigma_0/\{P \vee Q\}_K))$ occur in $NExt(\Sigma_1/\{C\}_I)\theta$ and $NExt(\Sigma_2/\{C\}_I)\theta$ in $NExt(\Pi)$ where θ is the substitution. Also, θ does not instantiate any element of $Rv(A)$. Also, $proj(0)(NExt(\Sigma_0/\{P \vee Q\}_K))$ is used in the decision procedure of $NExt(\Pi)$. Then the proof of this case will be finished immediately. Other cases are similar. ∎

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

Theorem 3: $NExt$ procedure and projection
Let A be a formula and Π be its proof. Also, let I be a declaration to Π and $Mark(I,\Pi)$ be as in theorem 2. Then, if $Mark(I,\Pi)$ succeeds, that is, if the marking condition is satisfied for any induction step proof contained in Π,
$NExt(Mark(I,\Pi)) = proj(I)(Ext(\Pi))$ holds.

Proof: By induction on the construction of the proof tree. Assume

$$NExt(Mark(I_i,\Pi_i)) = proj(I_i)(Ext(\Pi_i)) \qquad (i = 1,\cdots,k)$$

in the following proof tree

$$\Pi \overset{\text{def}}{=} \frac{\Pi_1 \cdots \Pi_k}{A}(R)$$

where I_i is the marking of Π_i. In the following, the marked version of Σ is also denoted Σ for simplicity.

1) Case $\Pi = \dfrac{\dfrac{\Sigma_0}{\exists x.A(x)} \quad \dfrac{\overset{[x,A(x)]}{\Sigma_1}}{C}}{C}(\exists\text{-}E)$:

$NExt(Mark(I,\Pi)) = NExt\left(\dfrac{Mark(\Sigma_0/\{\exists x.A(x)\}_K) \quad Mark(\Sigma_1/\{C\}_I)}{\{C\}_I}(\exists\text{-}E)\right)$

$= NExt(Mark(\Sigma_1/\{C\}_I))\theta$

where

$\theta \overset{\text{def}}{=} \left\{\begin{array}{c} proj(L)(Rv(A(x)))/tseq(1)(NExt(Mark(\Sigma_0/\{\exists x.A(x)\}_K))), \\ x/proj(0)(NExt(Mark(\Sigma_0/\{\exists x.A(x)\}_K))) \end{array}\right\} \quad (if\ 0 \in K)$

$\theta \overset{\text{def}}{=} \{proj(L)(Rv(A(x)))/NExt(Mark(\Sigma_0/\{\exists x.A(x)\}_K))\} \quad (if\ 0 \notin K)$

where L is the union of the markings of all the occurrence of $A(x)$ as hypothesis. On the other hand, by the induction hypothesis,

$NExt(Mark(\Sigma_0/\{\exists x.A(x)\}_K)) = proj(K)(Ext(\Sigma_0/\exists x.A(x)))$ and $K = L+1$ (if $0 \notin K$) or $\{0\} \cup (L+1)$. Therefore,

$\theta = \left\{\begin{array}{c} proj(L)(Rv(A(x)))/proj(L)(tseq(1)(Ext(\Sigma_0/\exists x.A(x)))), \\ x/proj(0)(Ext(\Sigma_0/\exists x.A(x))) \end{array}\right\} \quad (if\ 0 \in K)$

$\theta = \{proj(L)(Rv(A(x)))/proj(L)(tseq(1)(Ext(\Sigma_0/\exists x.A(x))))\} \quad (if\ 0 \notin K)$
Then, by lemma 3,
$= NExt(Mark(\Sigma_1/\{C\}_I))\theta_1$
where $\theta_1 \overset{\text{def}}{=} \{Rv(A(x))/tseq(1)(Ext(\Sigma_0/\exists x.A(x))), x/proj(0)(Ext(\Sigma_0/\exists x.A(x)))\}$
Then, by the induction hypothesis,
$= (proj(I)(Ext(\Sigma_1/C)))\theta_1 = proj(I)((Ext(\Sigma_1/C))\theta_1) = proj(I)(Ext(\Pi))$

2) Case $\Pi = \dfrac{\dfrac{\Sigma}{A}}{A \vee B}(\vee\text{-}I)_0$:
case $0 \in I$:
$NExt(Mark(I,\Pi)) = NExt\left(\dfrac{Mark(\Sigma/\{A\}_{(I-1)(<l(A))})}{\{A \vee B\}_I}(\vee\text{-}I)_0\right)$
$= (left, NExt(Mark(\Sigma/\{A\}_{(I-1)(<l(A))})), any[k])$
where $k = |I| - (1 + |(I-1)(<l(A))|)$.
Then, by the induction hypothesis,
$= (left, proj((I-1)(<l(A)))(Ext(\Sigma/A)), any[k])$
$= proj(\{0\} \cup ((I-1)(<l(A))+1) \cup I(>l(A)))(left, Ext(\Sigma/A), any[l(B)])$
$= proj(I)(Ext(\Pi))$

case $0 \notin I$:
$NExt(Mark(I,\Pi)) = NExt\left(\dfrac{Mark(\Sigma/\{A\}_{(I-1)(<l(A))})}{\{A \vee B\}_I}(\vee\text{-}I)_0\right)$
$= (NExt(Mark(\Sigma/\{A\}_{(I-1)(<l(A))})), any[l])$
where $l = |I| - |(I-1)(<l(A))|$.
Then, by the induction hypothesis,
$= (proj((I-1)(<l(A)))(Ext(\Sigma/A)), any[l])$
$= proj(((I-1)(<l(A))+1) \cup I(>l(A)))(left, Ext(\Sigma/A), any[l(B)])$
$= proj(I)(Ext(\Pi))$

3) Case $\Pi = \dfrac{\dfrac{\Sigma_0}{A \vee B} \quad \dfrac{\overset{[A]}{\Sigma_1}}{C} \quad \dfrac{\overset{[B]}{\Sigma_2}}{C}}{C}(\vee\text{-}E)$:
$NExt(Mark(I,\Pi))$
$= NExt\left(\dfrac{Mark(\Sigma_0/\{A \vee B\}_K) \quad Mark(\Sigma_1/\{C\}_I) \quad Mark(\Sigma_2/\{C\}_I)}{\{C\}_I}(\vee\text{-}E)\right)$
$= if\ left = proj(0)(NExt(Mark(\Sigma_0/\{A \vee B\}_K)))$
$\quad then\ NExt(Mark(\Sigma_1/\{C\}_I))\theta\ else\ NExt(Mark(\Sigma_2/\{C\}_I))\theta$
where
$$\theta \overset{\text{def}}{=} \left\{\begin{array}{l} proj(J_1)(Rv(A))/ttseq(1,|J_1|)(NExt(Mark(\Sigma_0/\{A \vee B\}_K))), \\ proj(J_2)(Rv(B))/tseq(|J_1|+1)(NExt(Mark(\Sigma_0/\{A \vee B\}_K))) \end{array}\right\}$$
where J_1 and J_2 are the unions of the markings of all the occurrences of A and B as hypotheses. On the other hand, by the induction hypothesis, $NExt(Mark(\Sigma_0/\{A \vee B\}_K)) =$

$proj(K)(Ext(\Sigma_0/A \vee B))$, and $K = \{0\} \cup (J_1 + 1) \cup (J_2 + 1 + l(A))$. Therefore,

$$\theta = \left\{ \begin{array}{l} proj(J_1)(Rv(A))/proj(J_1)(ttseq(1, l(A))(Ext(\Sigma_0/A \vee B))), \\ proj(J_2)(Rv(B))/proj(J_2)(tseq(l(A)+1)(Ext(\Sigma_0/A \vee B))) \end{array} \right\}$$

Then, by lemma 3,

$= if\ left = proj(0)(NExt(Mark(\Sigma_0/\{A \vee B\}_K)))$

$\qquad then\ NExt(Mark(\Sigma_1/\{C\}_I))\theta_1\ else\ NExt(Mark(\Sigma_2/\{C\}_I))\theta_1$

where

$$\theta_1 \stackrel{\text{def}}{=} \left\{ \begin{array}{l} Rv(A)/ttseq(1, l(A))(Ext(\Sigma_0/A \vee B)), \\ Rv(B)/tseq(l(A)+1)(Ext(\Sigma_0/A \vee B)) \end{array} \right\}$$

Then, by the induction hypothesis,

$= if\ left = proj(0)(proj(K)(Ext(\Sigma_0/A \vee B)))$

$\qquad then\ (proj(I)(Ext(\Sigma_1/C)))\theta_1\ else\ (proj(I)(Ext(\Sigma_2/C)))\theta_1$

As $0 \notin K$,

$= if\ left = proj(0)(Ext(\Sigma_0/A \vee B))$

$\qquad then\ proj(I)(Ext(\Sigma_1/C)\theta_1\ )else\ proj(I)(Ext(\Sigma_2/C)\theta_1)$

$= proj(I)(Ext(\Pi))$

In the case of modified $\vee$ code, $left = proj(0)\cdots$ part of the *if-then-else* construct is changed to A, and the proof is similar.

4) Case $\Pi = \dfrac{\dfrac{\Sigma_0}{A(0)} \quad \dfrac{\overset{[x > 0, A(x-1)]}{\Sigma_1}}{A(x)}}{\forall x.A(x)}(nat\text{-}ind)$:

$$NExt(Mark(I, \Pi)) = NExt\left(\frac{Mark\left(\dfrac{\Sigma_0}{\{A(0)\}_I}\right) \quad Mark\left(\dfrac{\Sigma_1}{\{A(x)\}_I}\right)}{\{\forall x.A(x)\}_I}(nat\text{-}ind) \right)$$

$$= \mu\overline{z}.\lambda x.if\ x = 0\ then\ NExt\left(Mark\left(\frac{\Sigma_0}{\{A(0)\}_I}\right)\right)\ else\ NExt\left(Mark\left(\frac{\Sigma_1}{\{A(x)\}_I}\right)\right)\theta$$

where $\theta \stackrel{\text{def}}{=} \{proj(I)(Rv(A(x-1)))/\overline{z}(x-1)\}$ and $\overline{z}$ is a sequence of new variables of length $|I|$. By lemma 3, $(NExt(Mark((\Sigma_1/\{A(x)\}_I))))\theta = (NExt(Mark((\Sigma_1/\{A(x)\}_I))))\theta_1$ where $\theta_1 \stackrel{\text{def}}{=} \{Rv(A(x-1))/\overline{z}'(x-1)\}$ and $\overline{z}'$ is a sequence of new variables of length $l(A(x))$ such that $proj(I)(\overline{z}') = \overline{z}$. Then, by the induction hypothesis,

$= \mu\overline{z}.\lambda x.if\ x = 0\ then\ proj(I)(Ext(\Sigma_0/A(0)))\ else\ proj(I)(Ext(\Sigma_1/A(x)))\theta_1$

$= \mu\overline{z}.\ proj(I)(\lambda x.\ if\ x = 0\ then\ Ext(\Sigma_0/A(0))\ else\ Ext(\Sigma_1/A(x))\theta_1)$

$= \mu\overline{z}.\ proj(I)(P)$

where $P \stackrel{\text{def}}{=} \lambda x.\ if\ x = 0\ then\ Ext(\Sigma_0/A(0))\ else\ Ext(\Sigma_1/A(x))\theta_1$. Assume that P is expanded to $(M_0, \cdots, M_{n-1})$ $(n = l(A(0)) = l(A(x)))$. Then,

$$\mu\overline{z}'.P = (f_0, \cdots, f_{n-1})$$

where $f_i = \mu z_i.N_i$, $\overline{z}' = (z_0, \cdots, z_{n-1})$ and N_i is obtained from M_i by substituting other f_js to free occurrences of z_js $(j \neq i)$ as explained in 2.1. Let $I = \{i_1, \cdots, i_m\}$, then

$\mu\overline{z}.\ proj(I)(P) = \mu\overline{z}.(M_{i_1},\cdots,M_{i_m})$.
Note that by lemma 3, any variable, z, such that $z \in \overline{z}'$ and $z \notin \overline{z}$ does not occur in M_{i_p} $(1 \leq p \leq m)$. Therefore, f_k $(k \notin I)$ does not occur in f_{i_p} $(1 \leq p \leq m)$. Hence, $\mu\overline{z}.(M_{i_1},\cdots,M_{i_m}) = (f_{i_1},\cdots,f_{i_m}) = proj(I)(\mu\overline{z}'.\ P) = proj(I)(Ext(\Pi))$

7) Other cases are similar or easy. ∎

7. Example

Here, the example of a prime number checker program is investigated.

7.1 Extraction of a Prime Number Checker Program by *Ext*

The specification of the program which takes any natural number as input and returns the boolean value, T, when the given number is prime, otherwise returns F is as follows:
Specification

$$\begin{aligned}\forall p:nat.\ \ (p \geq 2 \supset \exists b:bool.\ (\ &(\forall d:nat.\ (1 < d < p \supset \neg(d \mid p)) \wedge b = T)\\ &\vee(\exists d:nat.\ (1 < d < p \wedge (d \mid p)) \wedge b = F)))\end{aligned}$$

where $(x \mid y) \overset{\mathrm{def}}{=} \exists z.y = x \cdot z$.

This specification can be proved by using the following lemma:
Lemma: $\forall p:nat.\ \forall z:nat.\ (z \geq 2 \supset A(p,z))$
where

$$A(p,z) \overset{\mathrm{def}}{=} \exists b:bool.\ (P_0(p,z,b) \vee P_1(p,z,b))$$

$$P_0(p,z,b) \overset{\mathrm{def}}{=} \forall d:nat.\ (1 < d < z \supset \neg(d \mid p)) \wedge b = T$$

$$P_1(p,z,b) \overset{\mathrm{def}}{=} \exists d:nat.\ (1 < d < z \wedge (d \mid p)) \wedge b = F$$

Proof of specification

$$\dfrac{\dfrac{[p:nat] \qquad \dfrac{[p:nat] \qquad \dfrac{\begin{matrix}[p:nat]\\ \Sigma\end{matrix}}{\forall p:nat.\ \forall z:nat.\ (z \geq 2 \supset A(p,z))}(\textbf{Lemma})}{\forall z:nat.\ (z \geq 2 \supset A(p,z))}(\forall\text{-}E)}{p \geq 2 \supset A(p,p)}(\forall\text{-}E)}{\forall p:nat.(p \geq 2 \supset A(p,p))}(\forall\text{-}I)$$

The proof of the lemma, which will be denoted Π_{Len} in the following, is given in the **Appendix**, and the program extracted by *Ext* is as follows:

$prime \overset{\text{def}}{=} \lambda p.\ Ext(\Pi_{Len})(p)(p)$

```
$Ext(\Pi_{Len})$
$\overset{\text{def}}{=} \lambda p.\ \mu(z_0, z_1, z_2, z_3).$
        $\lambda z.\ if\ z = 0$
        $then\ any[4]$
        $else\ \ if\ z = 1$
              $then\ any[4]$
              $else\ \ if\ z = 2$
                    $then\ (T, left, any[2])$
                    $else\ \ if\ proj(0)((z_1, z_2, z_3)(z-1)) = left$
                          $then\ \ if\ proj(0)(Ext(\mathbf{prop})(p)(z-1)) = left$
                                $then\ (T, left, any[2])$
                                $else\ (F, right, z-1,\ tseq(1)(Ext(\mathbf{prop})(p)(z-1)))$
                          $else\ (F, right, z_2(z-1), z_3(z-1))$
```

$Ext(\mathbf{prop})$ is a function which takes natural numbers, m and n, as input and returns $(right, d)$ if m can be divided by n ($m = d \cdot n$) and $(left, any[1])$ otherwise.
$Ext(\Pi_{Len})$ is a multi-valued recursive call function which calculates a sequence of terms of length four. The boolean value which denotes whether the given number is prime is the 0th element of the sequence.

7.2 Program Extraction by Declaration, Marking and $NExt$

(1) Declaration
The realizing variables of the specification are sequence of variables of length four: (w_0, w_1, w_2, w_3). w_0,w_1,w_2,and w_3 are the variable for $\exists$ symbol on $b : bool$, the variable for $\vee$ symbol which connects P_0 and P_1, the variable for $\exists$ symbol on $d : nat$, and the variable for $\exists$ symbol in $(d \mid p)$ respectively.
As the only information needed here is the value of b, w_0 should be specified, that is, the declaration is $\{0\}$.

(2) Marking and Backtracking
It turns out that the marked proof tree, which is obtained with the declaration $\{0\}$ and $Mark$, does not satisfy the marking condition. The main part of the proof of the lemma is performed in mathematical induction. The marking of the conclusion of the induction proof is $\{0\}$, and the marking of an occurrence of the induction hypothesis (actually the induction hypothesis occurs only once in the proof) is $\{1\}(\not\subset \{0\})$. Therefore, $Mark$ fails. Then, the declaration is enlarged to $\{0,1\}$ and the marking procedure is performed again. The marking of the occurrence of the induction hypothesis is $\{1\}$ this time, and the marking condition is satisfied. Then, $NExt$ is ready to extract the program.

(3) Program Extraction by $NExt$

The $NExt$ procedure extracts the following program from the marked proof tree obtained in (2).

$$prime' = \lambda p.\ NExt(Mark(\Pi_{Len}))(p)(p)$$

```
$NExt(Mark(\Pi_{Len}))$
$= \lambda p.\ \mu(z_0, z_1).$
        $\lambda z.\ if\ z = 0$
        $then\ any[2]$
        $else\ \ if\ z = 1$
                $then\ any[2]$
                $else\ \ if\ z = 2$
                        $then\ (T, left)$
                        $else\ \ if\ z_1(z-1) = left$        $\cdots (*)$
                                $then\ \ if\ proj(0)(Ext(\mathbf{prop})(p)(z-1)) = left$
                                        $then\ (T, left)$
                                        $else\ (F, right)$
                                $else\ (F, right)$
```

Comparing the above code with $Ext(\Pi_{Len})$, the reason why the declaration should be $\{0,1\}$ (not $\{0\}$) is as follows: To calculate the boolean value which indicates whether the input natural number is prime, information as to whether the input can be divided by a natural number less than the input is necessary. That information is given as the 1st code, $lcft$ or $right$, of the term sequence calculated by the main loop of the multi-valued recursive call function.

Note that only the 1st element of the sequence is calculated at the recursive call step (see $(*)$ part). This is what the marking of the induction hypothesis, $\{1\}$, means.

(4) Alternative Extraction

The extracted program will be more efficient if the whole proof is normalized. In fact, $red_{\forall}$ can be applied to the proof of the specification. If the declaration is $\{1\}$, a program which returns the constants, $left$ and $right$, instead of boolean values is extracted. The same program can also be extracted by changing the specification to the following and give the declaration, $\{0\}$.

$$\forall p : nat.\ (p \geq 2 \supset ((\forall d : nat.\ (1 < d < p \supset \neg(d \mid p))) \vee (\exists d : nat.\ (1 < d < p \wedge (d \mid p)))))$$

7.3 Proof Tree Analysis

By using the proof theoretic characterization of critical applications explained in 4.3, the reason why the declaration should be enlarged to $\{0,1\}$ in the previous subsection can be explained in terms of the structure of the marked proof tree.

7.3.1 Main Paths from Induction Hypothesis

The main part of the proof of the lemma is performed in mathematical induction, and Figure 1 is the skeleton of the proof tree of the induction step. This is a part of the proof tree which is a collection of the formula occurrences along the main paths from an occurrence of the induction hypothesis which actually occurs only once in the proof. The formula occurrences in Figure 1 with the index number, (1), (2), $\cdots$, can be found in the proof tree in **Appendix** with the same index numbers. The discharged hypotheses of some of ($\supset$-I) and ($\vee$-E) applications are not shown in the figure because they are not along the main paths.

Formulas A to F are in the following form:

$A(z) = z \geq 2 \supset A(p,z) = * \supset B(z)$

$B(z) = \exists b.P_0(p,z,b) \vee P_1(p,z,b) = \exists b.C(z,b)$

$C(z,B) = P_0(p,z,B) \vee P_1(p,z,B) = D_0(z,B) \vee D_1(z,B)$

$D_0(z,B) = (\forall d.(1 < d < z \supset \neg(d|p)) \wedge B = T) = E_0(z) \wedge *$

$D_1(z,B) = (\exists d.(1 < d < z \wedge (d|p)) \wedge B = F) = E_1(z) \wedge *$

$E_0(z) = \forall d.(1 < d < z \supset \neg(d|p)) = \forall d.F_0(z)$

$E_1(z) = \exists d.(1 < d < z \wedge (d|p)) = \exists d.F_1(z)$

$F_0(z) = 1 < d < z \supset \neg(d|p) = * \supset G_0$

$F_1(z) = 1 < d < z \wedge (d|p) = G_1(z) \wedge G_2$

$G_0 = \neg(d|p) \quad G_1(z) = 1 < d < z \quad G_2 = (d|p)$

where $*$ is the abbreviation of some particular formula. $C(z,b)$, $D_0(z,b)$, and $D_1(z,b)$ are abbreviated to $C(z)$, $D_0(z)$, and $D_1(z)$.

There are three main paths from the occurrence of the induction hypothesis, $A(x-1)$:

$S_0 \overset{\text{def}}{=}$ (1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(11),(12),(13),(14),(15),(16),(17),(18),(19)

$S_1 \overset{\text{def}}{=}$ (1),(2),(3),(20),(21),(22),(23),(24),(25),(26),(27),(28),(29),(30),(15),(16),(17),(18),(19)

and $S_2 \overset{\text{def}}{=}$ (1),(2),(3),(20),(21),(31),(32),(25),(26),(27),(28),(29),(30),(15),(16),(17),(18),(19).

There are five non-trivial segments along S_0, S_1 and S_2:

(a) (7),(8) (b) (13),(14),(15),(16),(17) (c) (30),(15),(16),(17) (d) (18),(19) (e) (26),(27).

Segments (b) and (c) will be critical after the marking.

7.3.2 Initial Marking

The marked proof tree initiated by the declaration, $\{0\}$, is given in Figure 2.

After the marking, the non-trivial segments, (b) and (c), become proper segments. Also, because the major premises of the ($\exists$-E) and ($\vee$-E) applications, (2) and (3), are along the main paths, S_1, S_2 and S_3, (b) and (c) are critical segments. The indispensable marking number of the occurrence of $B(x-1)$ is 1, so that the marking of the induction hypothesis contains 1 which is not contained in the declaration, $\{0\}$.

7.3.3 Re-marking

The marking of the induction hypothesis is $\{1\}$ ($\not\subseteq \{0\}$), so that the declaration is enlarged to $\{0, 1\}$. Perform the marking again to obtain the marked proof tree is given in Figure 3. In the marked proof tree of Figure 3, the marking number, 1 ,of the formula occurrence indexed by (3) is the indispensable marking number, but it is contained in the declaration.

8. Conclusion

A method to extract redundancy-free realizer codes from constructive proofs was presented in this paper. The method allows fine grained specification of redundancy, and most of the analysis of redundancy is performed automatically. The set notation in the Nuprl and ITT by Gödeborg group and $\Diamond$-bounded formulas in PX are also the notations to specify the redundancy. For example, by transforming a specification, $\forall x.\exists y.\exists z.\exists w.A(x,y,z,w)$, to $\forall x.\exists y.\exists z.\Diamond\exists w.A(x,y,z,w)$, a function that calculates the values of y and z can be extracted in PX. However, a new proof must be given when the specification is changed. Also, if a function that calculates only the values of y and w are needed, $\Diamond$-notation cannot handle it. The set notation is similar in this respect. On the other hand, one should just declare $\{0, 2\}$ to the specification in the method presented in the paper.

Paulin-Mohring's version of the Calculus of Constructions also allows fine grained specification of redundancy. Her idea is to make a copy of the calculus with the constant *Prop* replaced by a new constant *Spec*, and the theorems and proofs are described in a mixture of the original calculus and the copy of it. The program extraction is performed only on the copy of the calculus. Our method uses a system of notations called declaration and marking instead of a copy of the original formal system. The basic idea is to perform the program analysis of redundancy at proof level, and the metalogical system of notations is sufficient for the analysis. The analysis of redundancy is performed by the marking procedure which may fail if the marking condition is not satisfied. However, the marking condition can be satisfied by implementing the backtracking mechanism given in Section 4.

The proof theoretic characterization of the marking condition given in Section 4 is not satisfactory. To give an equivalent condition to the marking condition in the proof theoretic notions would be an interesting research theme in the future.

REFERENCES

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

[Beeson 85] Beeson, M., "*Foundation of Constructive Mathematics*", Springer, 1985

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

[Coquand 88] Coquand, T. and Huet, G., "The Calculus of Constructions", Information and Computation 76, 1988

[Goad 80] Goad, C.A., "*Computational Uses of the Manipulation of Formal Proofs*", Ph.D. Thesis, Stanford University, 1980

[Hayashi 88] Hayashi, S. and Nakano, H., "*PX - A programming logic*", The MIT Press, 1988

[Howard 80] Howard, W. A., "The Formulae-as-types Notion of Construction", in *'Essays on Combinatory Logic, Lambda Calculus and Formalism'*, Eds J. P. Seldin and J. R. Hindley, Academic Press, 1980

[Nordström 83] Nordström, B. and Petersson, K., "Types and specifications", *Information Processing 83*, North-Holland, 1983

[Paulin-Mohring 89] Paulin-Mohring, C., "Extracting F^{ω}'s Programs from Proofs in the Calculus of Constructions", *Conference Record of the 16th Annual ACM Symposium on Principles of Programming Languages*, 1989

[Prawitz 65] Prawitz, D., "*Natural Deduction*", Almqvist & Wiksell, 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 87] Takayama, Y., "Writing Programs as QJ-Proofs and Compiling into PROLOG Programs", *Proceedings of 4th Symposium on Logic Programming*, 1987

[Takayama 88] Takayama, Y., "**QPC**: QJ-Based Proof Compiler – Simple Examples and Analysis –", *Proceedings of 2nd European Symposium on Programming*, LNCS 300, 1988

[Troelstra 73] Troelstra, A. S., "*Mathematical investigations of intuitionistic arithmetic and analysis*", Springer Lecture Notes in Mathematics, Vol. 344, 1973

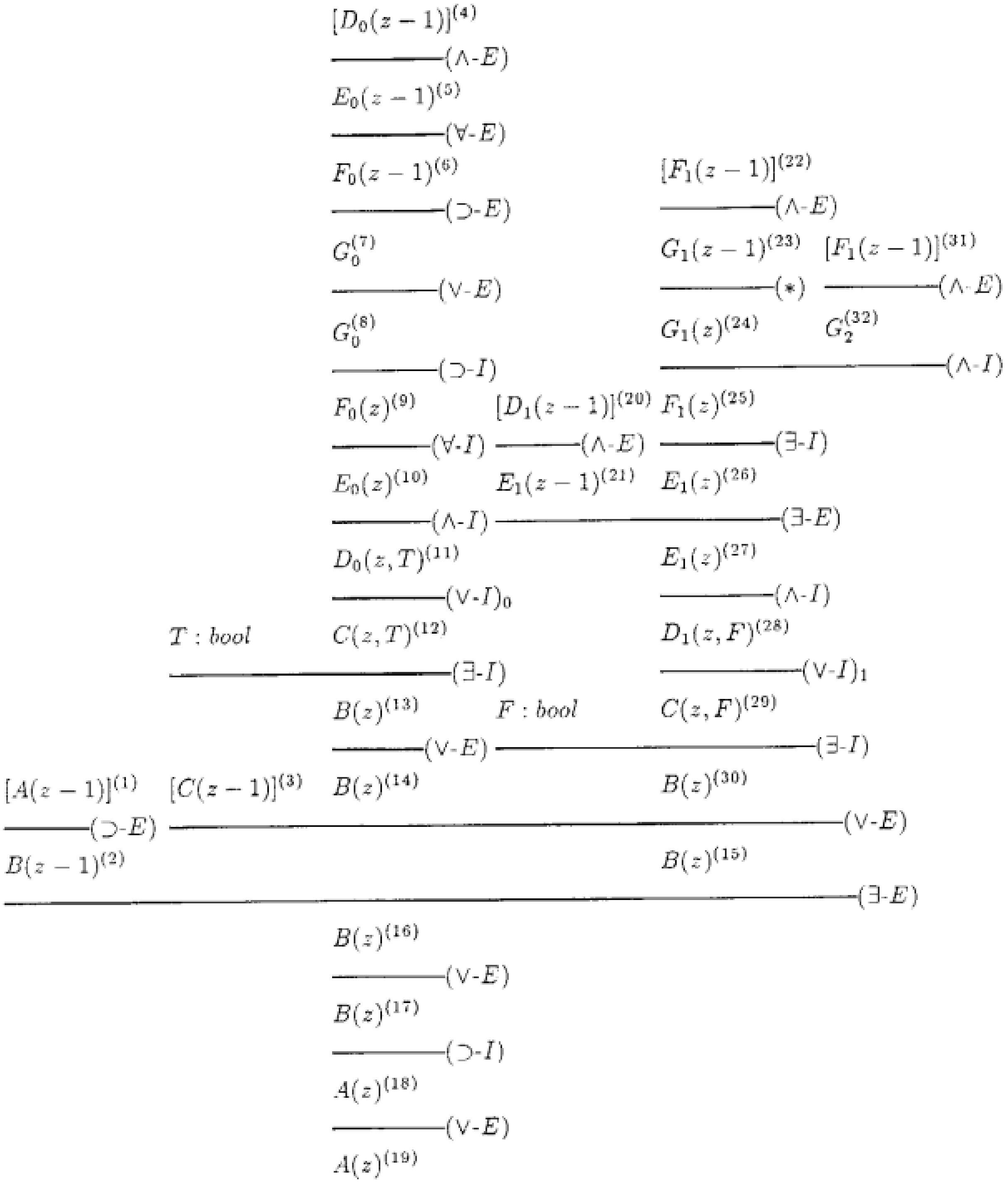

Figure 1

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[A(z-1)]_{\{1\}}}{B(z-1)_{\{1\}}}(\supset\text{-}E) \quad \dfrac{[C(z-1)]_{\{0\}} \quad \dfrac{\dfrac{T : bool_{\{0\}} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[D_0(z-1)]_\phi}{E_0(z-1)_\phi}(\wedge\text{-}E)}{F_0(z-1)_\phi}(\forall\text{-}E)}{G_{0\phi}}(\supset\text{-}E)}{G_{0\phi}}(\vee\text{-}E)}{F_0(z)_\phi}(\supset\text{-}I)}{E_0(z)_\phi}(\forall\text{-}I)}{D_0(z,T)_\phi}(\wedge\text{-}I)}{C(z,T)_\phi}(\vee\text{-}I)_0}{B(z)_{\{0\}}}(\exists\text{-}I)}{B(z)_{\{0\}}}(\vee\text{-}E) \quad \dfrac{F : bool_{\{0\}} \quad \dfrac{\dfrac{\dfrac{\dfrac{[D_1(z-1)]_\phi}{E_1(z-1)_\phi}(\wedge\text{-}E) \quad \dfrac{\dfrac{\dfrac{\dfrac{[F_1(z-1)]_\phi}{G_1(z-1)_\phi}(\wedge\text{-}E)}{G_1(z)_\phi}(*) \quad \dfrac{[F_1(z-1)]_\phi}{G_{2\phi}}(\wedge\text{-}E)}{F_1(z)_\phi}(\wedge\text{-}I)}{E_1(z)_\phi}(\exists\text{-}I)}{E_1(z)_\phi}(\exists\text{-}E)}{D_1(z,F)_\phi}(\wedge\text{-}I)}{C(z,F)_\phi}(\vee\text{-}I)_1}{B(z)_{\{0\}}}(\exists\text{-}I)}{B(z)_{\{0\}}}(\vee\text{-}E)}{B(z)_{\{0\}}}(\exists\text{-}E)}{B(z)_{\{0\}}}(\vee\text{-}E)}{A(z)_{\{0\}}}(\supset\text{-}I)}{A(z)_{\{0\}}}(\vee\text{-}E)$$

Figure 2

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[A(z-1)]_{\{1\}}}{B(z-1)_{\{1\}}}(\supset\text{-}E) \qquad \dfrac{[C(z-1)]_{\{0\}} \quad \dfrac{\dfrac{T : bool_{\{0\}} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[D_0(z-1)]_\phi}{E_0(z-1)_\phi}(\land\text{-}E)}{F_0(z-1)_\phi}(\forall\text{-}E)}{G_{0\phi}}(\supset\text{-}E)}{G_{0\phi}}(\lor\text{-}E)}{F_0(z)_\phi}(\supset\text{-}I)}{E_0(z)_\phi}(\forall\text{-}I)}{D_0(z,T)_\phi}(\land\text{-}I)}{C(z,T)_{\{0\}}}(\lor\text{-}I)_0}{B(z)_{\{0,1\}}}(\exists\text{-}I)}{B(z)_{\{0,1\}}}(\lor\text{-}E) \quad \dfrac{F : bool_{\{0\}} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[D_1(z-1)]_\phi}{E_1(z-1)_\phi}(\land\text{-}E) \quad \dfrac{\dfrac{\dfrac{\dfrac{[F_1(z-1)]_\phi}{G_1(z-1)_\phi}(\land\text{-}E)}{G_1(z)_\phi}(*) \quad \dfrac{[F_1(z-1)]_\phi}{G_{2\phi}}(\land\text{-}E)}{F_1(z)_\phi}(\land\text{-}I)}{E_1(z)_\phi}(\exists\text{-}I)}{E_1(z)_\phi}(\exists\text{-}E)}{D_1(z,F)_\phi}(\land\text{-}I)}{C(z,F)_{\{0\}}}(\lor\text{-}I)_1}{B(z)_{\{0,1\}}}(\exists\text{-}I)}{B(z)_{\{0,1\}}}(\lor\text{-}E)}{B(z)_{\{0,1\}}}(\exists\text{-}E)}{B(z)_{\{0,1\}}}(\lor\text{-}E)}{A(z)_{\{0,1\}}}(\supset\text{-}I)}{A(z)_{\{0,1\}}}(\lor\text{-}E)
$$

Figure 3

Appendix: Proof of Lemma (Π_{Len})

Main Proof

$$\dfrac{\dfrac{\dfrac{\begin{matrix}[p]\\ \Sigma_0\end{matrix}}{0 \geq 2 \supset A(p,0)} \qquad \dfrac{\begin{matrix}[p,\ z-1,\ z-1 \geq 2 \supset A(p,z-1)]\\ \Sigma_1\end{matrix}}{z \geq 2 \supset A(p,z)}}{\forall z.\ (z \geq 2 \supset A(p,z))}(nat\text{-}ind)}{\forall p.\ \forall z.\ (z \geq 2 \supset A(p,z))}(\forall\text{-}I)$$

Extracted Code by *Ext*:

$$\begin{array}{l}\lambda p.\ \mu(z_0, z_1, z_2, z_3).\ \lambda z.\\ \qquad if\ z = 0\ then\ Ext(\Sigma_0/0 \geq 2 \supset A(p,0))\ else\ Ext(\Sigma_1/z \geq 2 \supset A(p,z))\sigma_0\end{array}$$

where $\sigma_0 \overset{\text{def}}{=} \{Rv(z-1 \geq 2 \supset A(p,z-1))/(z_0, z_1, z_2, z_3)(z-1)\}$

- Proof of $p \vdash 0 \geq 2 \supset A(p,0)$ $\quad(\Sigma_0)$

$$\dfrac{\dfrac{\dfrac{[0 \geq 2]}{\bot}(*)}{A(p,0)}(\bot\text{-}E)}{0 \geq 2 \supset A(p,0)}(\supset\text{-}I)$$

Extracted Code by *Ext*: *any*[4]

- Proof of $p,\ z-1,\ z-1 \geq 2 \supset A(p,z-1) \vdash z \geq 2 \supset A(p,z)$ $\quad(\Sigma_1)$

$$\dfrac{\dfrac{[z-1 : nat]}{z = 1 \vee z \geq 2}(*) \quad \dfrac{\dfrac{\dfrac{[z \geq 2] \quad [z = 1]}{\bot}(*)}{A(p,z)}(\bot\text{-}E)}{z \geq 2 \supset A(p,z)}(\supset\text{-}I) \quad \dfrac{\begin{matrix}[z \geq 2, p, z-1]\\ [z-1 \geq 2 \supset A(p,z-1)]\\ \Sigma_{11}\end{matrix}}{z \geq 2 \supset A(p,z)^{(18)}}}{z \geq 2 \supset A(p,z)^{(19)}}(\vee\text{-}E)$$

Extracted Code by *Ext* (modified $\vee$-code):
$if\ z = 1\ then\ any[4]\ else\ Ext(\Sigma_{11}/z \geq 2 \supset A(p,z))$

- Proof of $p, z-1, z \geq 2,\ z-1 \geq 2 \supset A(p,z-1) \vdash z \geq 2 \supset A(p,z)$ $\quad(\Sigma_{11})$

$$\dfrac{\dfrac{\dfrac{[z \geq 2]}{z = 2 \vee z \geq 3}(*) \quad \dfrac{\begin{matrix}[z = 2]\\ \Sigma_{110}\end{matrix}}{A(p,z)} \quad \dfrac{\begin{matrix}[z \geq 3, z-1]\\ [z-1 \geq 2 \supset A(p,z)]\\ \Sigma_{111}\end{matrix}}{A(p,z)^{(16)}}}{A(p,z)^{(17)}}(\vee\text{-}E)}{z \geq 2 \supset A(p,z)^{(18)}}(\supset\text{-}I)$$

Extracted Code by *Ext* (modified $\vee$ code):

$$if\ z = 2\ then\ Ext(\Sigma_{110}/A(p,z))\ else\ Ext(\Sigma_{111}/A(p,z))$$

- Proof of Σ_{110}

$$\dfrac{[z=2] \qquad \dfrac{\overline{T}^{(*)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[d:nat] \quad [1<d<2]}{\bot}(*)}{\neg(d \mid p)}(\bot\text{-}E)}{1<d<2 \supset \neg(d \mid p)}}{\forall d.\ (1<d<2 \supset \neg(d \mid p))} \quad \dfrac{\overline{T}^{(*)}}{T=T}}{P_0(p,2,T)}(\wedge\text{-}I)}{P_0(p,2,T) \vee P_1(p,2,T)}}{A(p,2)}}{A(p,z)}(=\text{-}E)$$

Extracted Code by Ext: $(T, left, any[2])$

- Proof of Σ_{111}

$$\dfrac{\dfrac{\dfrac{[z \geq 3]}{z-1 \geq 2}(*) \quad \begin{bmatrix} z-1 \geq 2 \\ \supset A(p,z-1) \end{bmatrix}^{(1)}}{\exists b.\ P_0(p,z-1,b) \vee P_1(p,z-1,b)^{(2)}}(\supset\text{-}E) \qquad \Pi_0}{A(p,z)^{(16)}}(\exists\text{-}E)$$

where Π_0 is as follows:

$$\dfrac{\begin{bmatrix} P_0(p,z-1,b) \\ \vee P_1(p,z-1,b) \end{bmatrix}^{(3)} \qquad \dfrac{\begin{matrix} [b, z \geq 3, z-1] \\ [P_0(p,z-1,b)] \\ \Sigma_{1110} \end{matrix}}{A(p,z)^{(14)}} \qquad \dfrac{\begin{matrix} [b, z-1] \\ [P_1(p,z-1,b)] \\ \Sigma_{1111} \end{matrix}}{A(p,z)^{(30)}}}{A(p,z)^{(15)}}(\vee\text{-}E)$$

Extracted Code by Ext:

$$\begin{pmatrix} if\ proj(0)(w_1, w_2, w_3) = left\ then\ Ext(\Sigma_{1110}/A(p,z)) \\ \qquad\qquad\qquad\qquad\qquad else\ Ext(\Sigma_{1111}/A(p,z)) \\ \\ \end{pmatrix} \sigma_1$$

where $(w_0, w_1, w_2, w_3) \stackrel{\text{def}}{=} Rv(z-1 \geq 2 \supset A(p,z-1))$ and $\sigma_1 \stackrel{\text{def}}{=} \{b/w_0, Rv(P_0(p,z-1,b) \vee P_1(p,z-1,b))/(w_1, w_2, w_3)\}$.

- Proof of Σ_{1110} : $b,\ z \geq 3,\ z-1,\ P_0(p,z-1,b) \vdash A(p,z)$

$$\dfrac{\Pi_1 \qquad \dfrac{\begin{matrix} [P_0(p,z-1,b)] \\ [\neg(z-1 \mid p)] \\ \Sigma_{11100} \end{matrix}}{A(p,z)^{(13)}} \qquad \dfrac{\begin{matrix} [z-1] \\ [z \geq 3] \\ [(z-1 \mid p)] \\ \Sigma_{11101} \end{matrix}}{A(p,z)}}{A(p,z)^{(14)}}(\vee\text{-}E)$$

where Π_1 is as follows:

$$\dfrac{[z-1:nat] \qquad \dfrac{[p] \quad \dfrac{}{\forall m.\,\forall n.\,\neg(n\mid m)\vee(n\mid m)}(\mathbf{Prop.})}{\forall n.\,\neg(n\mid p)\vee(n\mid p)}}{\neg(z-1\mid p)\vee(z-1\mid p)}$$

Extracted Code by Ext:

$$\begin{array}{r}if\ proj(0)(Ext(\mathbf{Prop.})(p)(z-1)) = left\ then\ Ext(\Sigma_{11100}/A(p,z))\\ else\ Ext(\Sigma_{11101}/A(p,z))\sigma_2\end{array}$$

where $\sigma_2 \stackrel{\text{def}}{=} \{Rv((z-1\mid p))/tseq(1)(Ext(\mathbf{Prop})(p)(z-1))\}$

The proof of **Prop** is skipped.

- Proof of $\Sigma_{11100}:\ \neg(z-1\mid p),\ P_0(p,z-1,b)\vdash A(p,z)$

$$\dfrac{\dfrac{}{T}^{(*)} \qquad \dfrac{\dfrac{\Pi_2 \quad \dfrac{\dfrac{}{T}^{(*)}}{T=T}}{P_0(p,z,T)^{(11)}}(\wedge\text{-}I)}{P_0(p,z,T)\vee P_1(p,z,T)^{(12)}}}{\exists b.\,P_0(p,z,b)\vee P_1(p,z,b)^{(13)}}$$

where Π_2 is as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{[1<d<z]}{\begin{array}{c}1<d<z-1\\ \vee d=z-1\end{array}} \quad \dfrac{\begin{bmatrix}1<d\\ <z-1\end{bmatrix} \quad \dfrac{[d] \quad \dfrac{[P_0(p,z-1,b)^{(4)}]}{\forall d.\,1<d<z-1\supset\neg(d\mid p)^{(5)}}(\wedge\text{-}E)}{1<d<z-1\supset\neg(d\mid p)^{(6)}}}{\neg(d\mid p)^{(7)}} \quad \dfrac{\begin{array}{c}[d=z-1]\\ [\neg(z-1\mid p)]\end{array}}{\neg(d\mid p)}}{\neg(d\mid p)^{(8)}}}{1<d<z\supset\neg(d\mid p)^{(9)}}(\supset\text{-}I)}{\forall d.\,1<d<z\supset\neg(d\mid p)^{(10)}}(\forall\text{-}I)$$

Extracted Code by Ext: $(T, left, any[2])$

- Proof of $\Sigma_{11101}:\ z-1,\ z\geq 3,\ (z-1\mid p),\ \vdash A(p,z)$

$$\dfrac{\dfrac{}{F}^{(*)} \qquad \dfrac{\dfrac{\dfrac{[z-1:nat] \quad \dfrac{\dfrac{[z-1][z\geq 3]}{1<z-1<z} \quad [(z-1\mid p)]}{1<z-1<z\wedge(z-1\mid p)}}{\exists d.\,1<d<z\wedge(d\mid p)} \qquad \dfrac{\dfrac{}{F}^{(*)}}{F=F}}{P_1(p,z,F)}}{P_0(p,z,F)\vee P_1(p,z,F)}}{\exists b.\,P_0(p,z,b)\vee P_1(p,z,b)}$$

Extracted Code by Ext: $(F, right, z-1, Rv((z-1\mid p))$

- Proof of $b,\ z-1,\ P_1(p,z-1,b)\vdash A(p,z)\quad(\Sigma_{1111})$

$$\dfrac{\dfrac{}{F}(*) \qquad \dfrac{\dfrac{\dfrac{\dfrac{[P_1(p,z-1,b)^{(20)}]}{\exists d.\ 1<d<z-1\wedge(d\mid p)^{(21)}}(\wedge\text{-}E) \qquad \Pi_3}{\exists d.\ 1<d<z\wedge(d\mid p)^{(27)}}(\exists\text{-}E) \qquad \dfrac{\overline{F}}{F=F}}{P_1(p,z,F)^{(28)}}}{F_0(p,z,F)\vee P_1(p,z,F)^{(29)}}}{\exists b.\ P_0(p,z,b)\vee P_1(p,z,b)^{(30)}}$$

where Π_3 is as follows:

$$\dfrac{[d:nat] \qquad \dfrac{\dfrac{\dfrac{\begin{bmatrix}1<d\\ <z-1\\ \wedge(d\mid p)\end{bmatrix}^{(22)}}{1<d<z-1^{(23)}}(\wedge\text{-}E) \quad \dfrac{[z-1:nat]}{z-1<z}(*)}{1<d<z^{(24)}}(*) \qquad \dfrac{\begin{bmatrix}1<d\\ <z-1\\ \wedge(d\mid p)\end{bmatrix}^{(31)}}{(d\mid p)^{(32)}}}{1<d<z\wedge(d\mid p)^{(25)}}}{\exists d.\ 1<d<z\wedge(d\mid p)^{(26)}}$$

Extracted Code by Ext: $(F, right, (d, Rv((d\mid p)))\sigma_3)$ where $\sigma_3\overset{\mathrm{def}}{=}\{d/w_2, Rv((d\mid p))/w_3\}$