ICOT Technical Report: TR-395

TR-395

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

by

Y. Takayama

June, 1988

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

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

Yukihide Takayama

Institute for New Generation Computer Technology
1-4-28, Mita, 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 proof theoretic method to eliminate redundancy by analysing 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 technique to resolve critical cases. The method is studied in a simple constructive logic with primitive types, mathematical induction and its standard 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 proofs of theorems is thought to be one good approach to automated programming and program verification [Constable 86] [Takayama 87]. The automated theorem proving technique and language design are the key techniques to realize this paradigm. Executable codes can be extracted from constructive proofs by using the Curry-Howard isomorphism of formulae-as-types [Howard 80], or equivalently, the notion of realizability [Kleene 45] [Beeson 85]. This is also a key technique to make mathematics run on computers. Here, it raises the problem of extracting efficient codes from proofs, or, in other words, optimization at proof level.

A technique to optimize programs at proof level, *pruning*, is given in [Goad 80]. Generally, proofs contain a lot of information about the programs that correspond to the proofs, and the pruning technique uses the information in optimization drastically changing the strategies of algorithms. Goad also investigated an application of the proof normalization method to partial evaluation of proofs and a program extraction technique other than those using realizability. [Bates 79] applied a traditional syntactical optimization technique on the code extracted from proofs which might destroy the clear correspondence between proofs and program via realizability. [Sasaki 86] improved the program extraction algorithm based on realizability so that the trivial code for formulae that have no computational meaning can be simplified. The basic idea is as follows: if A and B are atomic formulae, then the computational meaning is trivial, so that the code extracted from, for instance, $A \wedge B$ is $(trivial, trivial)$. The modified program extractor simplifies the code to $trivial$. A similar technique is used in the PX system [Hayashi

86] as *type* 0 *formulae*. The QPC system [Takayama 88] uses a similar technique to Sasaki's, a normalization method to eliminate β-redex in the extracted codes, and the *modified* $\vee$ *code* technique to simplify some classes of decision procedures. However, the code extracted from constructive proofs still has redundancy, and it causes heavy runtime overhead.
If 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, the 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:

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

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 determined by $A(x, t(x))$, $t(x)$ denotes a term which contains a free variable, x, and $(term_1, term_2, \cdots)$ means the sequence of terms. In this paper, the executable codes extracted from constructive proofs, which are called *realizer codes* are in the form of sequences of terms or functions which output a sequence of terms. The codes contain 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 idea to overcome this problem would be introducing suitable notation to specify which part of the proof is necessary in terms of computation. The set notation, $\{x : A|B\}$, is introduced in the Nuprl system [Constable 86] as a weaker notion of $\exists x : A.\ B$. This is done to skip the extraction of the justification for B. [Mohring-Paulin 88] modified the *calculus of construction* [Coquand 86][Huet 86][Huet 88] by introducing two kinds of constants, *Prop* and *Spec*, to distinguish the formulae in proofs whose computational meaning is not necessary. These works are performed in the type theoretic formulation of constructive logic in the style of Martin-Löf.
This paper presents a proof theoretic method in the style of D. Prawitz to perform the program analysis at proof tree level, and to generate a redundancy-free realizer code. The method of program analysis can be presented quite clearly and naturally if it is performed at the proof tree level because proofs are the logical description of programs and have a lot of information about the programs. In some cases, the redundancy can be removed easily by applying a projection function to the extracted code. However, the situation around the redundancy is a little more complicated, particularly when the program extraction is performed on proofs which use induction, in other words, when the recursive call program is extracted. It needs a slightly

more detailed program analysis to distinguish the redundancy from the algorithmically significant part of extracted programs. It is mainly because, as the realizer codes can be naturally expressed in the form of sequence of terms, recursive call programs which correspond to proofs in induction have the form of multi-valued recursive call functions. The projection function on the extracted codes can be extended to a procedure on proof trees, and in this case, the meaning of the procedure becomes clearer.

The formalism of proof description and programs used in this paper is basically the intuitionistic version of the Gentzen type of natural deduction [Prawitz 65] with some additional program constructs such as λ- expressions and *if-then-else* terms, simple type structures, and the recursive type structure introduced in [Sato 85]. However, the feature of type structure, inference rules and the program constructs are not stressed in this paper. This formalism is more like C. Goad's formulation of constructive logic [Goad 80] than type theoretic formulation such as the Nuprl and the calculus of construction. Standard q-realizability is used as the program extraction algorithm.

Section 2 gives a formulation of the constructive logic and the realizability interpretation used in this paper. Here, realizability is given as the proof compilation procedure which is an algorithmic version of q-realizability. The definition of declaration to a specification and the marking procedure are given in section 3. These two methods are the basic idea of pre-processing to extract redundancy-free realizer codes. The critical part of declaration and the marking procedure is investigated in section 4. A sort of soundness theorem of the marking procedure is given at the end of this section. Section 5 works on the proof of the theorem. The modified proof compilation algorithm, which takes a marked proof tree as input and returns a redundancy-free realizer code, is defined in section 6. An example, a prime number checker program, is worked out in section 7. Section 8 is the conclusion.

2. Simple Constructive Logic

The constructive logic used here is, roughly, an intuitionistic version of first order natural deduction with mathematical induction plus higher order equality and inequality. It is a sugared subset of Sato's theory, **QJ** [Sato 85] [Sato 86].

2.1 Expressions and Inference Rules

The definition of the formal language is given somewhat informally here. See [Sato 86] for a more formal definition.

(1) Types

1) *nat* (the set of natural numbers) and *bool* (boolean type) are the primitive types.
2) If σ_0 and σ_1 are types, then $\sigma_0 \to \sigma_1$ is also a type.
3) If $\sigma_0 \cdots \sigma_{n-1}$ are types, then $\sigma_0 \times \cdots \times \sigma_{n-1}$ is also a type.

These type structures are not used explicitly in this paper. The definition of the recursive type structure is not given for the same reason.

(2) Terms (program constructs)

- Variables x, y, $\cdots$, and sequences of variables $\overline{x}$, $\overline{y}$, $\cdots$.
- Lambda abstraction: $\lambda(x_0, \cdots, x_{n-1}).\ Term \quad (0 \leq n)$.

The following equivalent relations hold:

a) $\lambda(nil).\ Term \equiv Term$ *b)* $\lambda(x_0, \cdots, x_{n-1}).\ Term \equiv \lambda x_0. \cdots \lambda x_{n-1}.Term$

• Atoms

1) Elements of *nat*: $0, 1, 2, \cdots$

2) *nil* (element of nil sequence), *left* and *right*, T and F (boolean), $any[n]$ (denotes n sequence of any atoms: $any[0] \stackrel{\text{def}}{=} nil$)

3) constant that represents absurdity: $\bot$

• If A is an atomic formula and B and C are terms, then *if* A *then* B *else* C is a term.

• If x is a variable or a sequence of variables and A is a term, then $\lambda x.A$ is a term.

• Application: $a(b)(c)\cdots(d)$

For any term, a, $a(nil) \stackrel{\text{def}}{=} a$.

• Sequences

If $t_0, \cdots, t_{n-1}$ are terms, then the sequence of terms, $(t_0, \cdots, t_{n-1})$, or simply $t_0, \cdots, t_{n-1}$, is a term.

1) If $n = 1$, the sequence (t_0) is equal to t_0.

2) Nil sequence: If $n = 0$, the sequence is denoted (nil).

3) Concatenation: The concatenation of two sequences, S_0 and S_1, is denoted (S_0, S_1). This notation will be used more generally: $(S_0, S_1, \cdots, S_{k-1})$ denotes the concatenation of k sequences. Note that $(S_0, (nil)) = (S_0, nil) = ((nil), S_0) = (nil, S_0) = S_0$ for any sequence, S_0.

4) Equivalence:

a) *if* A *then* $(a_0, \cdots, a_{n-1})$ *else* $(b_0, \cdots, b_{n-1})$

$\equiv$ (*if* A *then* a_0 *else* $b_0, \cdots,$ *if* A *then* a_{n-1} *else* b_{n-1})

b) $\lambda(x_0, \cdots, x_{n-1}).\ (a_0, \cdots, a_{m-1}) \equiv (\lambda(x_0, \cdots, x_{n-1}).\ a_0, \cdots, \lambda(x_0, \cdots, x_{n-1}).\ a_{m-1})$

• Fixed point operator μ.

(3) Formulae

1) $\bot$ is an atomic formula.

2) Equation and inequation of terms are atomic formulae.

Note that if t is a term and σ is a type, then $t : \sigma$ is an abbreviation of $t = t$.

3) If A and B are formulae, then $A \wedge B$, $A \vee B$ and $A \supset B$ are formulae.

4) If x is a variable of type σ and A is a formula, then $\exists x : \sigma.A$ and $\forall x : \sigma.A$ are formulae.

5) If A is a formula, $\neg A \stackrel{\text{def}}{=} A \supset \bot$ is a formula.

The type declarations of bound variables are often omitted. Also atomic formula $t : \sigma$ is often denoted simply t.

(5) Inference rules

• Introduction and elimination rules on $\wedge$, $\vee$, $\supset$, $\forall$ and $\exists$

• $\bot$ elimination rule

• Mathematical induction rule

• Rules on equality and inequality of terms

• Term construction rules

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

(6) Built-in functions

• *succ*, *pred* $\cdots$ successor and predecessor functions

• $proj(n)$ $\cdots$ function that projects the nth element of a sequence of terms

• $proj(I)$ $\cdots$ I is a finite set of natural numbers. If S is a sequence of terms of length n and $m < n$, then

$$proj(\{i_0, \cdots, i_m\}) \stackrel{\text{def}}{=} (proj(i_0)(S), \cdots, proj(i_m)(S))$$

• $tseq(n)$ ⋯ function that returns the subsequence of a given sequence: if S is a sequence of length n, then

$$tseq(i) = (proj(i)(S), proj(i+1)(S), \cdots, proj(n-1)(S))$$

• $ttseq(n, m)$ ⋯ function that returns the subsequence of a given sequence: if S is a sequence of length n, then

$$ttseq(i, l) = (proj(i)(S), proj(i+1)(S), \cdots, proj(i+(l-1))(S))$$

2.2 Proof Theoretic Terminology and Notation

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

Definition: it Principal sign & C-formula
(1) Let A be a formula that is not atomic. Then, A has exactly 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 $\exists$, respectively, is called the *principal sign* of A.
(2) A formula with the principal sign, C, is called the *C formula.*

Definition: Application & node
In a proof tree as follows

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

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

Definition: Subtree
If A is a formula occurrence in 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

$$\cfrac{\cfrac{[A]}{\Sigma}}{\cfrac{B}{C}}$$

is given, the subtree determined by B should be denoted $([A]/\Sigma/B)$. However, it is often referred to as $theproof(tree)\Sigma$ in the following description.

Definition: 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 formulae of $\Pi_0, \Pi_1, \cdots, \Pi_{n-1}$ respectively. Then, A_i is said to be *side-connected* with A_j $(0 \leq i, j < n)$.

Definition: Top & end-formula
(1) A *top-formula* in a formula 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: Minor & major-premise
In the following rules, C, C_0, C_1 and C_2 are said to be *minor premise*. A premise that is not minor is a *major premise*.

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

$$\frac{A \vee B \quad \begin{array}{c}[A]\\ C_0\end{array} \quad \begin{array}{c}[B]\\ C_1\end{array}}{C_2}(\vee\text{-}E) \qquad C_0, C_1, C_2 \text{ are of the same form.}$$

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

Definition: Cut

- An application of ($\supset$-I) succeeded by an application of ($\supset$-E) is called *cut*.

$$\frac{\dfrac{\Sigma_0}{B} \quad \dfrac{\begin{array}{c}[B]\\ \Sigma_1\end{array}}{B \supset A}(\supset\text{-}I)}{A}(\supset\text{-}E)$$

2.3 Realizing Variables Sequence and Length of Formulae

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

Definition: $Rv(A)$

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

Example:
$Rv(\forall x : nat.\ ((x \geq 0) \supset (x = 0 \vee \exists y : nat.\ succ(y) = x))) = (z_0, z_1)$
where z_0 denotes the information that shows which subformula of $\vee$ formula holds and z_1 denotes the realizing variables of $\exists y : nat.\ succ(y) = x$. Note that $Rv(succ(y) = x) = (nil)$.

Definition: Length of formulae
$l(A)$, 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 **q**-realizability as seen in [Sato 85] and Chapter VII of [Beeson 85]. The realizability is reformulated here as the *Ext* procedure [Takayama 88] that takes proof trees as input and returns functional style programs as output. The realizer code extracted by *Ext* is in the form of a sequence of terms.
In the following description, 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 may be both a variable and a sequence of variables. When X_i is a sequence of variables, T_i must also be a sequence of terms. Application of a substitution, θ, to a term, T, is denoted $T\theta$.

(1) For the realizer codes of assumptions, the realizing variable sequences are used:

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

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

$$Ext\left(\frac{\Sigma}{A}(Rule)\right) \stackrel{\text{def}}{=} nil$$

where A is an atomic formula.

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

- $$Ext\left(\cfrac{\cfrac{\Sigma_0}{A_0} \cdots \cfrac{\Sigma_{n-1}}{A_{n-1}}}{A_0 \wedge \cdots \wedge A_{n-1}}(\wedge\text{-}I)\right) \stackrel{\text{def}}{=} (Ext\left(\frac{\Sigma_0}{A_0}\right), \cdots, Ext\left(\frac{\Sigma_{n-1}}{A_{n-1}}\right))$$
- $$Ext\left(\cfrac{\cfrac{\Sigma}{A_0 \wedge \cdots \wedge A_{n-1}}}{A_i}(\wedge\text{-}E)\right) \stackrel{\text{def}}{=} ttseq\left(\textstyle\sum_{k=0}^{i-1} l(A_k), l(A_i)\right)\left(Ext\left(\frac{\Sigma}{A_0 \wedge \cdots \wedge A_{n-1}}\right)\right)$$
- $$Ext\left(\cfrac{\cfrac{\Sigma}{A}}{A \vee B}(\vee\text{-}I)\right) \stackrel{\text{def}}{=} (left, Ext\left(\frac{\Sigma}{A}\right), any[l(B)])$$
- $$Ext\left(\cfrac{\cfrac{\Sigma}{B}}{A \vee B}(\vee\text{-}I)\right) \stackrel{\text{def}}{=} (right, any[l(A)], Ext\left(\frac{\Sigma}{B}\right))$$

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

$$Ext\left(\cfrac{\cfrac{\Sigma_0}{A \vee B} \quad \cfrac{\begin{matrix}[A]\\ \Sigma_1\end{matrix}}{C} \quad \cfrac{\begin{matrix}[B]\\ \Sigma_2\end{matrix}}{C}}{C}(\vee\text{-}E)\right)$$

is as follows:

a) $if\ A\ then\ Ext\left(\dfrac{\begin{matrix}[A]\\ \Sigma_1\end{matrix}}{C}\right)\ else\ Ext\left(\dfrac{\begin{matrix}[B]\\ \Sigma_2\end{matrix}}{C}\right)$ **[modified ∨ code]**

$\cdots$ when both A and B are equations or inequations of terms

b) $if\ left = proj(0)\left(Ext\left(\dfrac{\Sigma_0}{A \vee B}\right)\right)\ then\ Ext\left(\dfrac{\begin{matrix}[A]\\ \Sigma_1\end{matrix}}{C}\right)\theta\ else\ Ext\left(\dfrac{\begin{matrix}[B]\\ \Sigma_2\end{matrix}}{C}\right)\theta$

$\cdots$ otherwise

where

$$\theta \stackrel{\text{def}}{=} \left\{\begin{matrix} Rv(A)/ttseq(1, l(Rv(A)))\left(Ext\left(\dfrac{\Sigma_0}{A \vee B}\right)\right), \\ Rv(B)/tseq(l(Rv(A))+1)\left(Ext\left(\dfrac{\Sigma_1}{A \vee B}\right)\right) \end{matrix}\right\}$$

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

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

- $$Ext\left(\dfrac{\dfrac{\begin{matrix}[A]\\ \Sigma\end{matrix}}{B}}{A \supset B}(\supset\text{-}I)\right) \stackrel{\text{def}}{=} \lambda Rv(A).\ Ext\left(\dfrac{\begin{matrix}[A]\\ \Sigma\end{matrix}}{B}\right)$$

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

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

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

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

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

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

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

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

$$\bullet\ Ext\left(\begin{matrix}\Sigma\\ \dfrac{\bot}{A}(\bot\text{-}E)\end{matrix} \right) \stackrel{\mathrm{def}}{=} any[l(A)].$$

(9) The multi-valued recursive call function is extracted by mathematical induction.

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

$$\stackrel{\mathrm{def}}{=} \mu\ \overline{z}.\ \lambda\ x.\ if\ x=0\ then\ Ext\left(\dfrac{\Sigma_0}{A(0)}\right)\ else\ Ext\left(\begin{matrix}[x:nat, A(x)]\\ \dfrac{\Sigma_1}{A(succ(x))}\end{matrix} \right)\sigma$$

where $\overline{z} = Rv(A(x))$, and $\sigma = \{\overline{z}/\overline{z}(pred(x)), x/pred(x)\}$.

Note that $\overline{z}$ denotes a sequence of variables, so that $\mu\overline{z}.\ \cdots$ is a multi-valued recursive call function. The multi-valued recursive call function of *degree* n, $\mu\ (z_0, \cdots, z_{n-1}).\ F(z_0, \cdots, z_{n-1})$ where $n \geq 1$ and $F(z_0, \cdots, z_{n-1})$ is a term with free variables, $z_0, \cdots, z_{n-1}$, is defined as follows:

1) Assume that $F(z_0, \cdots, z_{n-1})$ is equivalent to the following sequence of functions:

$$(F_0(z_0, \cdots, z_{n-1}), \cdots, F_{n-1}(z_0, \cdots, z_{n-1}))$$

2) Let $G_i \stackrel{\mathrm{def}}{=} \mu z_i.\ F_i(z_0, \cdots, z_{n-1})$, where $0 \leq i \leq n-1$.

3) Define H_i, where $0 \leq i \leq n-1$, inductively as follows:

(a) $H_0 \stackrel{\mathrm{def}}{=} G_0$

(b) For $1 \leq i \leq n-1$, let $H_i \stackrel{\mathrm{def}}{=} G_i\{z_0/H_0, \cdots, z_{n-1}/H_{n-1}\}$.

(c) Redefine H_k, where $0 \leq k \leq i$, as follows: $H'_k \stackrel{\mathrm{def}}{=} H_k\{z_i/G_i\}$

4) $\mu(z_0, z_1, z_2).F(z_0, \cdots, z_{n-1}) \stackrel{\mathrm{def}}{=} (H_0(z_0, \cdots, z_{n-1}), \cdots, H_{n-1}(z_0, \cdots, z_{n-1}))$.

Example:

Let $F(z_0, z_1, z_2) \stackrel{\mathrm{def}}{=} (\lambda x.\ p(z_0, z_1, z_2),\ \lambda y.\ q(z_0, z_1, z_2),\ \lambda z.\ r(z_0, z_1, z_2))$. By the definition and $(\mu\text{-}=)$ rule:

$$\frac{\mu z.\ F(z)}{\mu z.\ F(z) = F(\mu z.\ F(z))}(\mu\text{-}=)$$

$\mu(z_0, z_1, z_2).F(z_0, z_1, z_2) = (H_0(z_0, z_1, z_2), H_1(z_0, z_1, z_2), H_2(z_0, z_1, z_2))$
where

$$H_0 \stackrel{\text{def}}{=} \mu z_0.\lambda x.p(z_0,\ \mu z_1.\lambda y.q(z_0, z_1, \mu z_2.\lambda z.r(z_0, z_1, z_2)),\ \mu z_2.\lambda z.r(z_0, \mu z_1.\lambda y.q(z_0, z_1, z_2), z_2))$$

$$H_1 \stackrel{\text{def}}{=} \mu z_1.\lambda y.q(\mu z_0.\lambda x.p(z_0, z_1, \mu z_2.\lambda z.r(z_0, z_1, z_2)),\ z_1,\ \mu z_2.\lambda z.r(\mu z_0.\lambda x.p(z_0, z_1, z_2), z_1, z_2))$$

$$H_2 \stackrel{\text{def}}{=} \mu z_2.\lambda z.r(\mu z_0.\lambda x.p(z_0, \mu z_1.\lambda y.q(z_0, z_1, z_2), z_2),\ \mu z_1.\lambda y.q(\mu z_0.\lambda x.p(z_0, z_1, z_2), z_1, z_2),\ z_2)$$

The execution of multi-valued recursive functions is quite expensive, so that making the degree smaller is an effective way of generating an efficient realizer code.

Theorem 1 (Soundness of the Ext procedure):
Let A be a sentence. If $\vdash_{\mathbf{QPC}} A$ and P is its proof tree, then $\vdash_{\mathbf{QPC}} Ext(P)\ \mathbf{q}\ A$
where $a\ \mathbf{q}\ A$ means that a term, a, realizes the formula A.

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

3. Declaration and Marking of Proof Trees

The proof trees are a clear description of logical meaning of programs, so that the analysis to detect the redundancy of realizer codes is much easier to perform if it is performed at the proof tree level.
The realizer of a formula, A, is a sequence of codes of length $l(A)$ as defined 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 that it is necessary for end users to 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 should specify.
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 information is given in the end-formula, the information can be inherited 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 code that is not necessary. Consequently, end users may not specify the nodes in the proof tree about the redundancy; it suffices to specify them only in the conclusion of the proof.

3.1 Declaration to Specifications

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

Declaration indicates which values of the existentially quantified variables of a given theorem are needed. It is the only information that end users of the system need to specify: the other part is performed automatically. Suppose, for simplicity, that the given theorem is of the following canonical form:

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

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

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

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

The following restriction assures a sort of soundness, proved later.

Restriction: The marking numbers of a declarations cannot specify realizing variables of more than two subformulae of the specification which are separated by $\wedge$. For example, if the specification is of the form $A \wedge B$ and $l(A) = 2$ and $l(B) = 3$, marking such as $\{0, 3\}$ is thought to be illegal because 0 specifies a variable in $Rv(A)$ and 3 specifies a variable in $Rv(B)$.

3.2 Marking

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

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

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

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

The code extracted by q-realizability is

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

or equivalently

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

$Ext(A(x,s,t))$ denotes the code extracted from the subtree determined by $A(x,s,t)$. 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 ($\exists$-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 $\{1\}$. 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:

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

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

The marking procedure continues from the bottom of proof trees to the tops. The proof compilation procedure, Ext, should be modified to take marked proof trees as inputs and extract part of the 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

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

is the term which is the value of x bound by $\exists$. Let I be the marking of the conclusion, then t should be marked $\{0\}$ if $0 \in I$, otherwise the marking is ϕ. The marking of $A(t)$ is given as 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 of natural numbers, $K - 1 \stackrel{\text{def}}{=} \{a - 1 | a \in K\}$.

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:

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

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

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

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

$$\dfrac{[x] \quad P(x)}{\exists y.\ P(y)}(\exists\text{-}I)$$

and the marking of $[x]$ is $\{0\}$ so that x (variable) should be extracted from the proof tree determined by the minor premise, C. 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 Π_1, so that the marking is:

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

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

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

3.2.3 Marking of the $(\vee\text{-}E)$ Rule

The realizer code of C concluded by the following inference

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

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

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

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

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

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

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

J_0' and J_1' are obtained by translating J_0 and J_1 in the realizer sequence of $A \vee B$.

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

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

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

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

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

The marking of the subtree determined by A is as follows.
Case 1: The application of ($\supset$-E) is a Cut:
The realizer code of A as the minor premise is restricted by the marking of A as a hypothesis of the subtree determined by $A \supset B$. Let I be the marking of B, and let J be the union of the marking of As as a hypothesis:

$$\frac{\begin{matrix}\Pi_0 \\ A\end{matrix} \quad \dfrac{\begin{matrix}\{[A]\}_J \\ \Sigma_1 \\ \overline{B}\end{matrix}}{\{A \supset B\}_I} (\supset\text{-}I)}{\{B\}_I} (\supset\text{-}E)$$

Hence the marking of the subtree is:

$$\frac{\Sigma_0}{\{A\}_J}$$

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

3.2.5 Definition of the *Mark* Procedure

• Notational preliminary

Mark is defined in the following style:

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

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

$I + n \stackrel{\text{def}}{=} \{x + n \mid x + n \leq max(I), x \in I\}$
$I - n \stackrel{\text{def}}{=} \{x - n \mid x - n \geq 0, x \in I\}$
$I(< n) \stackrel{\text{def}}{=} \{x \in I \mid x < n\}$
$I(\geq n) \stackrel{\text{def}}{=} \{x \in I \mid x \geq n\}$

• Definition of *Mark*

$$Mark\left(\cfrac{t \quad \cfrac{\Sigma}{A(t)}}{\{\exists x.\ A(x)\}_I}(\exists\text{-}I)\right) \stackrel{\text{def}}{=} \begin{cases} \cfrac{\{t\}_\phi \quad Mark\left(\cfrac{\Sigma}{\{A(t)\}_{I-1}}\right)}{\{\exists x.\ A(x)\}_I}(\exists\text{-}I) & \text{if } 0 \notin I\ ; \\ \cfrac{\{t\}_{\{0\}} \quad Mark\left(\cfrac{\Sigma}{\{A(t)\}_{I-1}}\right)}{\{\exists x.\ A(x)\}_I}(\exists\text{-}I) & \text{otherwise.} \end{cases}$$

$$Mark\left(\cfrac{\cfrac{\Sigma_0}{\exists x.\ A(x)} \quad \begin{matrix}[t,\ A(t)]\\ \Sigma_1 \\ \hline C\end{matrix}}{\{C\}_I}(\exists\text{-}E)\right) \stackrel{\text{def}}{=} \cfrac{Mark\left(\cfrac{\Sigma_0}{\{\exists x.\ A(x)\}_K}\right) \quad Mark\left(\begin{matrix}[t,\ A(t)]\\ \Sigma_1 \\ \hline \{C\}_I\end{matrix}\right)}{\{C\}_I}(\exists\text{-}E)$$

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 maximal markings of hypotheses t and $A(t)$ obtained in

$$Mark\left(\begin{matrix}[t,\ A(t)]\\ \Sigma_1 \\ \hline \{C\}_I\end{matrix}\right)$$

$$Mark\left(\cfrac{\begin{matrix}[x:\sigma]\\ \Sigma \\ \hline A(x)\end{matrix}}{\{\forall x:\sigma.\ A(x)\}_I}(\forall\text{-}I)\right) \stackrel{\text{def}}{=} \cfrac{Mark\left(\begin{matrix}[x:\sigma]\\ \Sigma \\ \hline \{A(x)\}_I\end{matrix}\right)}{\{\forall x:\sigma.\ A(x)\}_I}(\forall\text{-}I)$$

$$Mark\left(\dfrac{t \quad \dfrac{\Sigma}{\forall x.\ A(x)}}{\{A(t)\}_I}(\forall\text{-}E)\right) \stackrel{\text{def}}{=} \dfrac{t \quad Mark\left(\dfrac{\Sigma}{\{\forall x.\ A(x)\}_I}\right)}{\{A(t)\}_I}(\forall\text{-}E)$$

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

Note that, according to the restriction on the declaration, at least one of $I(< l(A))$ and $I(\geq l(A)) - l(A)$ is ϕ.

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

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

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

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

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

$$\stackrel{\text{def}}{=} \begin{cases} \dfrac{Mark\left(\dfrac{\Sigma_0}{\{A \vee B\}_K}\right) \quad Mark\left(\dfrac{\begin{matrix}[A]\\ \Sigma_1\end{matrix}}{\{C\}_I}\right) \quad Mark\left(\dfrac{\begin{matrix}[B]\\ \Sigma_2\end{matrix}}{\{C\}_I}\right)}{\{C\}_I}(\vee\text{-}E) & \text{if } I \neq \phi \\ \dfrac{Mark\left(\dfrac{\Sigma_0}{\{A \vee B\}_\phi}\right) \quad Mark\left(\dfrac{\begin{matrix}[A]\\ \Sigma_1\end{matrix}}{\{C\}_\phi}\right) \quad Mark\left(\dfrac{\begin{matrix}[B]\\ \Sigma_2\end{matrix}}{\{C\}_\phi}\right)}{\{C\}_\phi}(\vee\text{-}E) & \end{cases}$$

where $K = \{0\} \cup (J_0 + 1) \cup (J_1 + l(A))$, and J_0 and J_1 are the maximal markings of $[A]$ and $[B]$.

$$Mark\left(\cfrac{\begin{array}{c}[A]\\ \Sigma\\ \hline B\end{array}}{\{A \supset B\}_I}(\supset\text{-}I)\right) \stackrel{\text{def}}{=} \cfrac{Mark\left(\begin{array}{c}[A]\\ \Sigma\\ \hline \{B\}_I\end{array}\right)}{\{A \supset B\}_I}(\supset\text{-}I)$$

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

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

$$\stackrel{\text{def}}{=} \cfrac{Mark\left(\cfrac{\Sigma_0}{\{A(0)\}_I}\right) \quad Mark\left(\cfrac{\begin{array}{c}[A(x)]\\ \Sigma_1\end{array}}{\{A(x+1)\}_I}\right)}{\{\forall x.\ A(x)\}_I}(nat\text{-}ind)$$

$$Mark\left(\cfrac{\cfrac{\Sigma_0}{x=y} \quad \cfrac{\Sigma_1}{A(x)}}{\{A(y)\}_I}(=\text{-}E)\right) \stackrel{\text{def}}{=} \cfrac{Mark\left(\cfrac{\Sigma_0}{x=y}\right) \quad Mark\left(\cfrac{\Sigma}{\{A(x)\}_I}\right)}{\{A(y)\}_I}(=\text{-}E)$$

$$Mark\left(\cfrac{\cfrac{\Sigma}{\bot}}{\{A\}_I}(\bot\text{-}E)\right) \stackrel{\text{def}}{=} \cfrac{Mark\left(\cfrac{\Sigma}{\{\bot\}_\phi}\right)}{\{A\}_I}(\bot\text{-}E)$$

$$Mark\left(\cfrac{\cfrac{\Sigma_0}{B_0} \cdots \cfrac{\Sigma_k}{B_k}}{\{A\}_\phi}(*)\right) \stackrel{\text{def}}{=} \cfrac{Mark\left(\cfrac{\Sigma_0}{\{B_0\}_\phi}\right) \cdots Mark\left(\cfrac{\Sigma_1}{\{B_k\}_\phi}\right)}{\{A\}_\phi}(*)$$

Termination pattern of $Mark$:

- Assumption

$Mark(\{[A]\}_I) \stackrel{\text{def}}{=} \{[A]\}_I$

- Trivial marking

$$Mark\left(\cfrac{B_0 \cdots B_k}{A}(*)\right) \stackrel{\text{def}}{=} \cfrac{B_0 \cdots B_k}{A}(*)$$

4. Critical Applications

4.1 Induction Hypothesis and Marking

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

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

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

The rest of this section is dedicated to an analysis of these critical applications of the rules.

4.2 Critical Segments

4.2.1 Problematic ($\vee$-E) and ($\exists$-E) Application

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

$$\cfrac{[\exists x.\ B(x) \vee C(x)] \qquad \cfrac{[B(x) \vee C(x)] \quad \cfrac{\begin{matrix}[x]\\ [B(x)]\\ \Sigma_0\end{matrix}}{A(x+1)} \quad \cfrac{\begin{matrix}[x]\\ [C(x)]\\ \Sigma_1\end{matrix}}{A(x+1)}}{A(x+1)}(\vee\text{-}E)}{A(x+1)}(\exists\text{-}E)$$

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

$$\cfrac{\{[\exists x.\ B(x) \vee C(x)]\}_L \qquad \cfrac{\{[B(x) \vee C(x)]\}_K \quad \cfrac{\begin{matrix}\{[x]\}_P\{[B(x)]\}_I\\ \Sigma_{00}\end{matrix}}{\{A(x+1)\}_{\{0\}}} \quad \cfrac{\begin{matrix}\{x\}_Q\{[C(x)]\}_J\\ \Sigma_{11}\end{matrix}}{\{A(x+1)\}_{\{0\}}}}{\{A(x+1)\}_{\{0\}}}(\vee\text{-}E)}{\{A(x+1)\}_{\{0\}}}(\exists\text{-}E)$$

where Σ_{00} and Σ_{11} are the suitably marked versions of Σ_0 and Σ_1. I and J are the union of the markings of $B(x)$ and $C(x)$, and P and Q are the union of the markings of x as hypotheses. Note that P and Q are either $\{0\}$ or ϕ.

Then K and L are as follows.

Case 1: $P \cup Q = \{0\}$

$$K = \{0\} \cup (I+1) \cup (J + l(B(x))$$

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

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

$$K = \{0\} \cup (I+1) \cup (J + l(B(x))$$
$$L = K + 1 = \{1\} \cup (I+2) \cup (J + l(B(x)) + 1)$$

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

4.2.2 Formal Definition of Critical Segments

Definition: Thread
Let $S \stackrel{\text{def}}{=} (A_1, A_2, \cdots, A_n)$ be a sequence of proof occurrences in a formula tree, Π. Then S is a *thread* iff
(1) A_1 is a top-formula in Π;
(2) A_i stands immediately above A_{i+1} in Π for each $i < n$;
(3) A_n is the end-formula of Π.

Definition: Segment
Let $S \stackrel{\text{def}}{=} (A_1, A_2, \cdots, A_n)$ be a sequence of consecutive formula occurrences in a thread in a proof tree Π. Then, S is a *segment* iff
(1) A_1 is not a conclusion of the application of ($\vee$-E) or ($\exists$-E);
(2) For arbitrary i ($< n$), A_i is a minor premise of an application of ($\vee$-E) or ($\exists$-E);
(3) A_n is not a minor premise of any application of ($\vee$-E) or ($\exists$-E).

Note that all formula occurrences in a segment are of the same form. Any formula occurrence, A, in a proof tree, Π, that is not a conclusion or a minor premise of the application of ($\vee$-E) or ($\exists$-E) is a segment by (1) and (3) of the definition. This kind of segment is called a *trivial segment* in the following description.

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

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

Definition: Path
Let $S \stackrel{\text{def}}{=} (A_1, A_2, \cdots, A_n)$ be a sequence in a deduction Π. S is a *path* iff
(1) A_1 is a top-formula in Π that is not discharged by an application of ($\vee$-E) and ($\exists$-E);
(2) A_i, for each $i < n$, is not the minor premise of an application of ($\supset$-E), and either (a) A_i is not the major premise of ($\vee$-E) or ($\exists$-E), and A_{i+1} is the formula occurrence immediately

below A_i, or (b) A_i is the major premise of an application of ($\vee$-E) or ($\exists$-E), and A_{i+1} is an assumption discharged by the application in Π;
(3) A_n is either a minor premise of ($\supset$-E), the end-formula of Π, or a major premise of an application of ($\vee$-E) or ($\exists$-E) such that no assumptions are discharged by the application.

Definition: Main path
The *main path* in a proof tree, Π, is the path whose last formula is the end-formula of Π.

Lemma: (Existence of indispensable marking)
Let $S \stackrel{\text{def}}{=} (A_1, A_2, \cdots, A_n)$ be a proper segment in a proof tree, and let Π be the subtree determined by A_n. Assume that there is a main path, P, from the major premise, F, attached to A_{n-1}, and let $S_0 = (B_1, B_2, \cdots, B_k)$, where $B_1 = F$, be the sequence of the formula occurrences along P such that B_j is a major premise attached to some A_{l_j} in S. Let $S_1 = (B_{i_1}, B_{i_2}, \cdots, B_{i_l})$ be a subsequence of S_0 such that B_{i_j} $(1 \le j \le l)$ is either (a) the major premise of the application of the ($\vee$-E) rule, or (b) the major premise of the application of the ($\exists$-E) rule such that at least one marking of the variable as the assumption of the application is not nil. Then, the marking of F contains the marking numbers, $\varphi(i_j)$ $(1 \le j \le l)$, where

$$\varphi(l) \stackrel{\text{def}}{=} \sum_{n=1}^{l} \psi(n)$$

$$\psi(n) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } n = 1 \\ l(A_0) & \text{if } B_{n-1} = A_0 \vee A_1 \text{ is a major premise of } (\vee\text{-}E) \text{ and } B_n = A_1 \\ 1 & \text{otherwise} \end{cases}$$

Proof: Let A_i and A_{i+1} be elements of S which are the minor premise and the conclusion of an application of ($\vee$-E) rule as follows. Assume that $A \vee B$ is an element of S_1.

$$\dfrac{\dfrac{\Sigma_0}{A \vee B} \quad \dfrac{\begin{matrix}[A]\\ \Sigma_1\end{matrix}}{A_i} \quad \dfrac{\begin{matrix}[B]\\ \Sigma_2\end{matrix}}{A_i'}}{\begin{matrix}A_{i+1}\\ \Pi_1\end{matrix}}(\vee\text{-}E) \qquad A_i, A_i' \text{ and } A_{i+1} \text{ have the same form.}$$

By the definition of $Mark$, the marking of $A \vee B$ contains 0 as a marking number.
Case 1: Assume that there is a formula occurrence of S in Π_1 (including A_{i+1}) which is a minor premise of an application of ($\exists$-E) and that the major premise, F_0, of the application precedes $A \vee B$ in S_0. Then, by the definition of $Mark$ for ($\exists$-E), the marking number, 0, of $A \vee B$ is incremented by 1 in the marking of F_0.
Case 2: Assume that there is a formula occurrence of S in Π_1 (including A_{i+1}) which is a minor premise of an application of ($\vee$-E) such that (a) the major premise, F_1, of the application precedes $A \vee B$ in S_0, and (b) $A \vee B$ stands on the left minor premise of the application. Then, by the definition of $Mark$ for ($\vee$-E), the marking number, 0, of $A \vee B$ is incremented by 1 in the marking of F_1.
Case 3: Assume that there is a formula occurrence of S in Π_1 (including A_{i+1}) which is a minor premise of an application of ($\vee$-E) such that (a) the major premise, F_2, of the application precedes $A \vee B$ in S_0, and (b) $A \vee B$ stands on the right minor premise of the application. Then, by the definition of $Mark$ for ($\vee$-E), the marking number, 0, of $A \vee B$ is incremented by $l(A)$ in the marking of F_2.

The proof is similar where A_i and A_{i+1} are the premise and the conclusion of an application of ($\exists$-E). The lemma follows from the above discussion. ∎

Note that the *Mark* procedure analyses the given proof tree from bottom to top along paths. Thus, according to the above lemma, if there is a formula occurrence which is a major premise attached to a formula occurrence in a proper segment in a main path from the induction hypothesis, such a formula occurrence may cause the problem illustrated in the previous section. When there is a path from the induction hypothesis to the conclusion of the induction step which is not a main path, another kind of problem is raised. This is discussed in the next section.

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

4.3 Critical ($\supset$-E) Applications

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

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

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

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

4.4 Critical ($\wedge$-$I\&E$) Applications

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

$$\dfrac{\dfrac{\dfrac{[A \wedge B]}{A}}{\begin{matrix}\Pi_0\\ A'\end{matrix}} \quad \dfrac{\Sigma_1}{B'}}{A' \wedge B'}$$

Let I be the non-nil marking of $A' \wedge B'$, and assume that $I(\geq l(A')) = I$. Then, the marking of A' is ϕ so that the marking of the induction hypothesis, $A \wedge B$, is also ϕ, i.e., different from I. This situation is problematic in terms of the correspondence of markings of induction hypotheses and conclusions of the induction steps explained in section 4.1.

4.5 Main Theorem

Definition: (*maximum segment*)
A *maximum segment* is a segment that begins with a consequence of an application of an I-rule or the ($\bot$-E) rule, and ends with a major premise of an E-rule.

Note that cut is a maximum segment.

Definition:
An application of ($\vee$-E) or ($\exists$-E) rule is said to be *redundant* iff it has a minor premise at which no assumption is discharged.

Definition: Normal deduction
A proof tree, Π, is *normal* iff
(1) Π contains no maximum segment, and
(2) Π contains no redundant applications of ($\vee$-E) or ($\exists$-E).

Theorem A: [Prawitz 65]
If $\Gamma \vdash A$ holds in the system for intuitionistic logic, then there is a normal deduction in this system of A from Γ.

For the normal proof trees, the soundness of the *Mark* procedure holds in the following sense.

Theorem 2:
Suppose that a formula, $\forall x.A(x)$, is proved by mathematical induction, and I is an arbitrary declaration of the conclusion. Let Π be a normal deduction of $A(x) \vdash A(x+1)$, and assume that there is no critical ($\wedge$-I&E) application in Π:

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

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

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

5. Proof of the Main Theorem

5.1 Form of Normal Proof Trees

Definition: Sequence of segments
Every path, π, can be obviously divided uniquely into consecutive segments (usually consisting of trivial segments): $\pi = \sigma_0, \cdots, \sigma_k$. This sequence is called the *sequence of segments in π*.

Theorem B: [Prawitz 65]
Let Π be a normal proof tree, let π be a path in Π, and let $\sigma_1, \sigma_2, \cdots, \sigma_n$ be the sequence of segments in π. Then there is a segment (minimum segment), σ_i, which separates two (possibly empty) parts of π, called the E-part and I-part of π, with the properties:
(1) For each σ_j in the E-part (i.e., $j < i$) it holds that σ_j is a major premise of an E-rule and that the formula occurring in σ_{j+1} is a subformula of the one occurring in σ_j;
(2) σ_i, provided that $i \neq n$, is a premise of an I-rule or of the ($\perp$-E) rule;
(3) For each σ_j in the I-part, except the last one, it holds that σ_j is a premise of an I-rule and that the formula occurring in σ_j is a subformula of the one occurring in σ_{j+1}.

Note that theorems A and B hold for pure intuitionistic natural deduction. Our system also has rules on (in)equalities and several other rules on terms. Those rules do not eliminate or introduce any logical constants. The sequence of premises and conclusions of these rules is similar to segments in this respect. Therefore, the minimum segment may not be a segment in the proofs in our system; it may be a sequence of formulae deduced by those inference rules. However, for simplicity, this sequence is also called a minimum segment.

5.2 Proof of Theorem 2

Let Π be a normalized proof tree from $A(x)$ (induction hypothesis) to $A(x+1)$ (conclusion of the induction step). If there is a path in Π that contains a minor premise of ($\supset$-E), the marking of $A(x)$ should be trivial. Therefore, assume here that all the paths in Π are main paths.
Let S be an arbitrary main path in Π. According to the theorem in the last section, there exists a segment, σ_i, that separates the segment sequence of the path into the E-part and I-part.
$A(x)$ and $A(x+1)$ are of the same form if the difference of parameters x and $x+1$ is neglected. Therefore, if $S_0 \stackrel{\text{def}}{=} C_1, C_2, \cdots, C_k$ is the sequence of logical constant occurrences that are eliminated in the E-part and $S_1 \stackrel{\text{def}}{=} C'_1, C'_2, \cdots, C'_l$ is the sequence of logical constant occurrences that are introduced in the I-part, and S_0 and S_1 are equal as multisets. Furthermore, the order of elimination of logical constants in the E-part and the reverse order of introducing of logical constants in the I-part is equal because the construction of $A(x)$ and $A(x+1)$ is the same, so that $S_1 = C_k, C_{k-1}, \cdots, C_1$. The theorem follows by mathematical induction on the length of S_0 (and equally S_1), k.
The base case is clear because, by the definition of *Mark*, the marking of minor premises of applications of ($\exists$-E) and ($\vee$-E) are equal to the conclusions. For the induction step, as there are no critical segments, it suffices to check the logical constants, $\forall$, $\supset$ and $\wedge$.

Case 1 ($\forall$):

$$
Mark\left(\begin{array}{c}
\dfrac{\bar{t}^{(*)} \quad [\forall x.A(x)]}{A(t)}(\forall\text{-}E) \\
\Pi \\
\dfrac{A'(s)}{\{\forall y.A'(y)\}_I}(\forall\text{-}I)
\end{array}\right)
$$

Π contains the minimum segment, and let Π' be the marked version of Π. $A(t)$ and $A'(s)$ are of the same construction and differ at most in some atomic formulae. By the induction hypothesis, the markings of $A(t)$ and $A'(s)$ are equal. Then,

$$
= \begin{array}{c}
Mark\left(\dfrac{\bar{t}^{(*)} \quad [\forall x.A(x)]}{\{A'(t)\}_I}(\forall\text{-}E)\right) \\
\Pi' \\
\dfrac{\{A'(s)\}_I}{\{\forall y.A'(y)\}_I}(\forall\text{-}I)
\end{array}
= \begin{array}{c}
\dfrac{\bar{t}^{(*)} \quad \{[\forall x.A(x)]\}_I}{\{A'(t)\}_I}(\forall\text{-}E) \\
\Pi' \\
\dfrac{\{A'(s)\}_I}{\{\forall y.A'(y)\}_I}(\forall\text{-}I)
\end{array}
$$

Consequently, the markings of $\forall x.A(x)$ and $\forall y.A'(y)$ are equal.

Case 2 ($\supset$):

$$
Mark\left(\begin{array}{c}
\dfrac{\dfrac{\Sigma_0}{A} \quad [A \supset B]}{B} \\
(\Pi_1, [A']) \\
\dfrac{B'}{\{A' \supset B'\}_I}(\supset\text{-}I)
\end{array}\right)
$$

Π_1 contains the minimum segment, and A, B and A', B' differ at most in some atomic subformulae. The markings of B and B' are equal by the induction hypothesis. Then,

$$
= \dfrac{\begin{array}{c}
Mark\left(\dfrac{\dfrac{\Sigma_0}{A} \quad [A \supset B]}{\{B\}_I}(\supset\text{-}E)\right) \\
(\Pi_1', \{[A']\}_J) \\
\{B'\}_I
\end{array}}{\{A' \supset B'\}_I}(\supset\text{-}I)
= \dfrac{\begin{array}{c}
\dfrac{\dfrac{\Sigma_0}{A} \quad \{[A \supset B]\}_I}{\{B\}_I}(\supset\text{-}E) \\
(\Pi_1', \{[A']\}_J) \\
\{B'\}_I
\end{array}}{\{A' \supset B'\}_I}(\supset\text{-}I)
$$

Consequently, the markings of $A \supset B$ and $A' \supset B'$ are equal.

Case 3 ($\wedge$):

- $(\wedge\text{-}E)_0$ rule

$$
Mark\left(\dfrac{\begin{array}{c}\dfrac{[A\wedge B]}{A}(\wedge\text{-}E)\\ \Pi_0\\ A'\end{array}\qquad \dfrac{\Sigma_1}{B'}}{\{A'\wedge B'\}_I}\right)=\dfrac{Mark\left(\begin{array}{c}\dfrac{[A\wedge B]}{A}(\wedge\text{-}E)\\ \Pi_0\\ \{A'\}_{I(<l(A))}\end{array}\right)\quad Mark\left(\dfrac{\Sigma_1}{\{B'\}_{I(\geq l(A))-l(A)}}\right)}{\{A'\wedge B'\}_I}
$$

By the induction hypothesis, the markings of A and A' are equal. Then,

$$
=\dfrac{\begin{array}{c}Mark\left(\dfrac{[A\wedge B]}{\{A\}_{I(<l(A))}}(\wedge\text{-}E)\right)\\ \Pi_0'\\ \{A'\}_{I(<l(A))}\end{array}\qquad \dfrac{\Sigma_1'}{\{B'\}_{I(\geq l(A))-l(A)}}}{\{A'\wedge B'\}_I}
$$

$$
=\dfrac{\begin{array}{c}\dfrac{\{[A\wedge B]\}_{I(<l(A))}}{\{A\}_{I(<l(A))}}(\wedge\text{-}E)\\ \Pi_0'\\ \{A'\}_{I(<l(A))}\end{array}\qquad \dfrac{\Sigma_1'}{\{B'\}_{I(\geq l(A))-l(A)}}}{\{A'\wedge B'\}_I}
$$

because there is no critical $(\wedge\text{-}I\&E)$ application, $I(<l(A))=I$.

- $(\wedge\text{-}E)_1$ rule

$$
Mark\left(\dfrac{\dfrac{\Sigma_0}{A'}\qquad \begin{array}{c}\dfrac{[A\wedge B]}{B}(\wedge\text{-}E)\\ \Pi_1\\ B'\end{array}}{\{A'\wedge B'\}_I}\right)
$$

$$
=\dfrac{Mark\left(\dfrac{\Sigma_0}{\{A'\}_{I(\geq l(A))-l(A)}}\right)\quad Mark\left(\begin{array}{c}\dfrac{[A\wedge B]}{B}(\wedge\text{-}E)\\ \Pi_1\\ \{B'\}_{I(\geq l(A))-l(A)}\end{array}\right)}{\{A'\wedge B'\}_I}
$$

By the induction hypothesis, the markings of A and A' are equal. Then,

$$
=\dfrac{\dfrac{\Sigma_0'}{\{A'\}_{I(<l(A))}}\qquad \begin{array}{c}Mark\left(\dfrac{[A\wedge B]}{\{B\}_{I(\geq l(A))-l(A)}}(\wedge\text{-}E)\right)\\ \Pi_1'\\ \{B'\}_{I(\geq l(A))-l(A)}\end{array}}{\{A'\wedge B'\}_I}=\dfrac{\dfrac{\Sigma_0'}{\{A'\}_{I(<l(A))}}\qquad \begin{array}{c}\dfrac{\{[A\wedge B]\}_{I(\geq l(A))}}{\{B\}_{I(\geq l(A))-l(A)}}(\wedge\text{-}E)\\ \Pi_1'\\ \{B'\}_{I(\geq l(A))-l(A)}\end{array}}{\{A'\wedge B'\}_I}
$$

because there is no critical ($\wedge$-I&E) application, $I(\geq l(A)) = I$.

∎

6. 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 analyse the subtree determined by A.
3) If the formula, A, is trivially marked, all the codes for $Rv(A)$ should be extracted.

The following is the definition of the modified version of the Ext procedure, $NExt$.

(1) Assumptions:
$NExt(\{[A]\}_I) \stackrel{\text{def}}{=} proj(I)(Rv(A))$

(2) Atomic formulae:

$$NExt\left(\frac{\{A_0\}_{J_0} \cdots \{A_k\}_{J_k}}{\{B\}_I}(Rule)\right) \stackrel{\text{def}}{=} nil$$

where B is an atomic formula

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

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

Note that if $I_i = \phi$, $NExt\left(\frac{\Sigma_1}{\{A_i\}_{I_i}}\right) = (nil) \quad i = 0 \cdots 1$.

$$\bullet\ NExt\left(\frac{\dfrac{\Sigma}{\{A_0 \wedge \cdots \wedge A_{n-1}\}_J}}{\{A_i\}_I}(\wedge\text{-}E)\right) \stackrel{\text{def}}{=} NExt\left(\frac{\Sigma}{\{A_0 \wedge \cdots \wedge A_{n-1}\}_J}\right)$$

where $i = 0 \cdots n-1$.

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

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

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

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

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

is as follows:

a) $if\ A\ then\ NExt\left(\cfrac{\begin{matrix}\{[A]\}_{J_1}\\ \Sigma_1\end{matrix}}{\{C\}_I}\right)\ else\ NExt\left(\cfrac{\begin{matrix}\{[B]\}_{J_2}\\ \Sigma_2\end{matrix}}{\{C\}_I}\right)$ **[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) $if\ left = proj(0)\left(NExt\left(\cfrac{\Sigma_0}{\{A\vee B\}_{J_0}}\right)\right)\ then\ NExt\left(\cfrac{\begin{matrix}\{[A]\}_{J_1}\\ \Sigma_1\end{matrix}}{\{C\}_I}\right)\theta\ else\ NExt\left(\cfrac{\begin{matrix}\{[B]\}_{J_2}\\ \Sigma_2\end{matrix}}{\{C\}_I}\right)\theta$

otherwise
J_0 must contain 0.

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

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

- $$NExt\left(\cfrac{\cfrac{\begin{matrix}[x:Type]\\ \Sigma\end{matrix}}{\{A(x)\}_I}}{\{\forall x:Type.\ A(x)\}_I}(\forall\text{-}I)\right) \stackrel{\text{def}}{=} \lambda x.\ NExt\left(\cfrac{\begin{matrix}[x:Type]\\ \Sigma\end{matrix}}{\{A(x)\}_I}\right)$$

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

(6) The code that is in the form of an function application is extracted from the proofs in ($\supset$-E) and ($\forall$-E): Note that proofs must be cur-free.

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

- $NExt\left(\dfrac{\overline{t:\sigma}^{(*)} \quad \dfrac{\Sigma}{\{\forall x:\sigma.\ A(x)\}_I}}{\{A(t)\}_I}(\forall\text{-}E)\right) \stackrel{\text{def}}{=} NExt\left(\dfrac{\Sigma}{\{\forall x:\sigma.\ A(x)\}_I}\right)(t)$

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

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

- $$NExt\left(\dfrac{\overline{\{\exists x:\sigma.\ A(x)\}_J}^{(*)} \quad \begin{array}{c}[\{x:\sigma\}_K, \{A(x)\}_L] \\ \Sigma \\ \hline \{C\}_I\end{array}}{C}(\exists\text{-}E)\right)$$

$$\stackrel{\text{def}}{=} NExt\left(\begin{array}{c}[\{x:\sigma\}_K, \{A(x)\}_L] \\ \Sigma \\ \hline \{C\}_I\end{array}\right)\theta$$

$$\text{where } \theta \stackrel{\text{def}}{=} \left\{\begin{array}{c} proj(L)(Rv(A(x)))/tseq(1)\left(NExt\left(\overline{\{\exists x:\sigma.\ A(x)\}_J}^{(*)}\right)\right), \\ x/proj(0)\left(NExt\left(\overline{\{\exists x:\sigma.\ A(x)\}_J}\right)\right) \end{array}\right\}.$$

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

- $NExt\left(\begin{array}{c}\Sigma \\ \hline \perp \\ \hline \{A\}_I\end{array}(\perp\text{-}E)\right) \stackrel{\text{def}}{=} any[k] \qquad \text{where } k = |I|$

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

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

$$\stackrel{\text{def}}{=} \mu\,\overline{z}.\ \lambda\,x.\ if\ x = 0\ then\ NExt\left(\dfrac{\Sigma_0}{\{A(0)\}_I}\right)\ else\ NExt\left(\begin{array}{c}[x:nat, \{A(x)\}_I] \\ \Sigma_1 \\ \hline \{A(succ(x))\}_I\end{array}\right)\sigma$$

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

(10) Trivial marking:

$NExt\left(\dfrac{A_0 \cdots A_k}{B}(Rule)\right) \stackrel{\text{def}}{=} Ext\left(\dfrac{A_0 \cdots A_k}{B}(Rule)\right)$

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

Theorem 3: Soundness of the $NExt$ procedure
Let A be a sentence and D be the declaration. If $\vdash_{\mathbf{QPC}} A$ and Π is its proof tree, then

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

Proof: Straightforward ▮

7. Example

Here, the example of a prime number checker program is investigated. The redundancy-free code is extracted by the method given in the previous sections.

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

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

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

$$A(p,z) \stackrel{\text{def}}{=} \exists b : nat.\ (P_0(p,z,b) \vee P_1(p,z,b))$$

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

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

Proof of specification

$$\dfrac{[p:nat] \qquad \dfrac{[p:nat] \qquad \dfrac{\dfrac{\Sigma}{\forall p : nat.\ \forall z : nat.\ (z \geq 2 \supset A(p,z))}\text{(Lemma)}}{}}{\forall z : nat.\ (z \geq 2 \supset A(p,z))}}{\dfrac{p \geq 2 \supset A(p,p)}{\forall p : nat.(p \geq 2 \supset A(p,p))}}$$

The proof of the lemma, Σ, is given in the **Appendix**, and the program extracted by Ext is as follows:

$prime \stackrel{\text{def}}{=} \lambda p.\ Ext(\Sigma)(p)(p)$

$$
\begin{array}{l}
Ext(\Sigma) \stackrel{\text{def}}{=} \lambda p.\ \mu(z_0, z_1, z_2, z_3). \\
\quad \lambda z.\ if\ z = 0 \\
\quad then\ any[4] \\
\quad else\ \ if\ z = 1 \\
\quad\quad then\ any[4] \\
\quad\quad else\ \ if\ z = 2 \\
\quad\quad\quad then\ (T, left, any[2]) \\
\quad\quad\quad else\ \ if\ proj(1)((z_0, z_1, z_2, z_3)(z-1)) = left\ (*) \\
\quad\quad\quad\quad then\ \ if\ proj(0)(Ext(\mathbf{prop})(p)(z-1)) = left \\
\quad\quad\quad\quad\quad then\ (T, left, any[2]) \\
\quad\quad\quad\quad\quad else\ (F, right, z-1,\ proj(1)(Ext(\mathbf{prop})(p)(z-1))) \\
\quad\quad\quad\quad else\ (F, right, z_2(z-1), z_3(z-1))
\end{array}
$$

$Ext(\mathbf{prop})$

$\stackrel{\text{def}}{=} \lambda m.\ \lambda n.\ \ (if\ proj(1)Ext(\mathbf{Th.}) = 0\ then\ \ (right, proj(0)Ext(\mathbf{Th.}))\ else\ left, any[1])$

$Ext(\mathbf{lemma})$ is a multi-valued recursive call function which calculates four sequences of terms. The boolean value which denotes whether the given number is prime is the first element of the sequence, so that the other part of the sequence seems to be redundant. However, the decision procedure (*) uses the second term of the sequence. This means that the second term of the sequence is also necessary. The other part, the third and fourth elements, is redundant.

7.2 Declaration

The realizing variables sequence of the specification is as follows:

$$(z_0, z_1, z_2, z_3)$$

where

$$z_0 \stackrel{\text{def}}{=} variable\ for\ \exists\ symbol\ on\ b : bool$$

$$z_1 \stackrel{\text{def}}{=} variable\ for\ \vee\ symbol\ which\ connects\ P_0\ and\ P_1$$

$$z_2 \stackrel{\text{def}}{=} variable\ for\ \exists\ symbol\ on\ d : nat$$

$$z_3 \stackrel{\text{def}}{=} variable\ for\ \exists\ symbol\ in\ (d \mid p)$$

Note that $(d \mid p) \stackrel{\text{def}}{=} \exists r : nat.\ p = r \cdot d$, so that $l((d \mid p)) = 1$.

As the only information needed is whether the given natural number is prime or not, z_0 should be specified, i.e., the declaration is $\{0\}$.

7.3 Proof Tree Analysis

7.3.1 Main Paths from Induction Hypothesis

The main part of the lemma is proved by mathematical induction, and Figure 1 is the skeleton of the proof tree of the induction step. This is a part of the proof tree involved in the paths from the induction hypothesis to the conclusion of the induction step. G_0 and G_1 are the formulae whose logical constants are not eliminated in the deduction. Formulae A to F are of the following form:

$A(z) = * \supset B(z)$

$B(z) = \exists b.C(z, b)$

$C(z) = D_0(z, b) \vee D_1(z, b)$

$C(z, Term) = D_0(z, Term) \vee D_1(z, Term)$

$D_0(z) = E_0(z) \wedge *$

$D_1(z) = E_0(z) \wedge *$

$E_0(z) = \forall d.F_0(z)$

$E_1(z) = \exists d.F_1(z)$

$F_0(z) = * \supset G_0(z)$

$F_1(z) = G_1(z) \wedge G_2(z)$

where $*$ is the abbreviation of some particular formula.

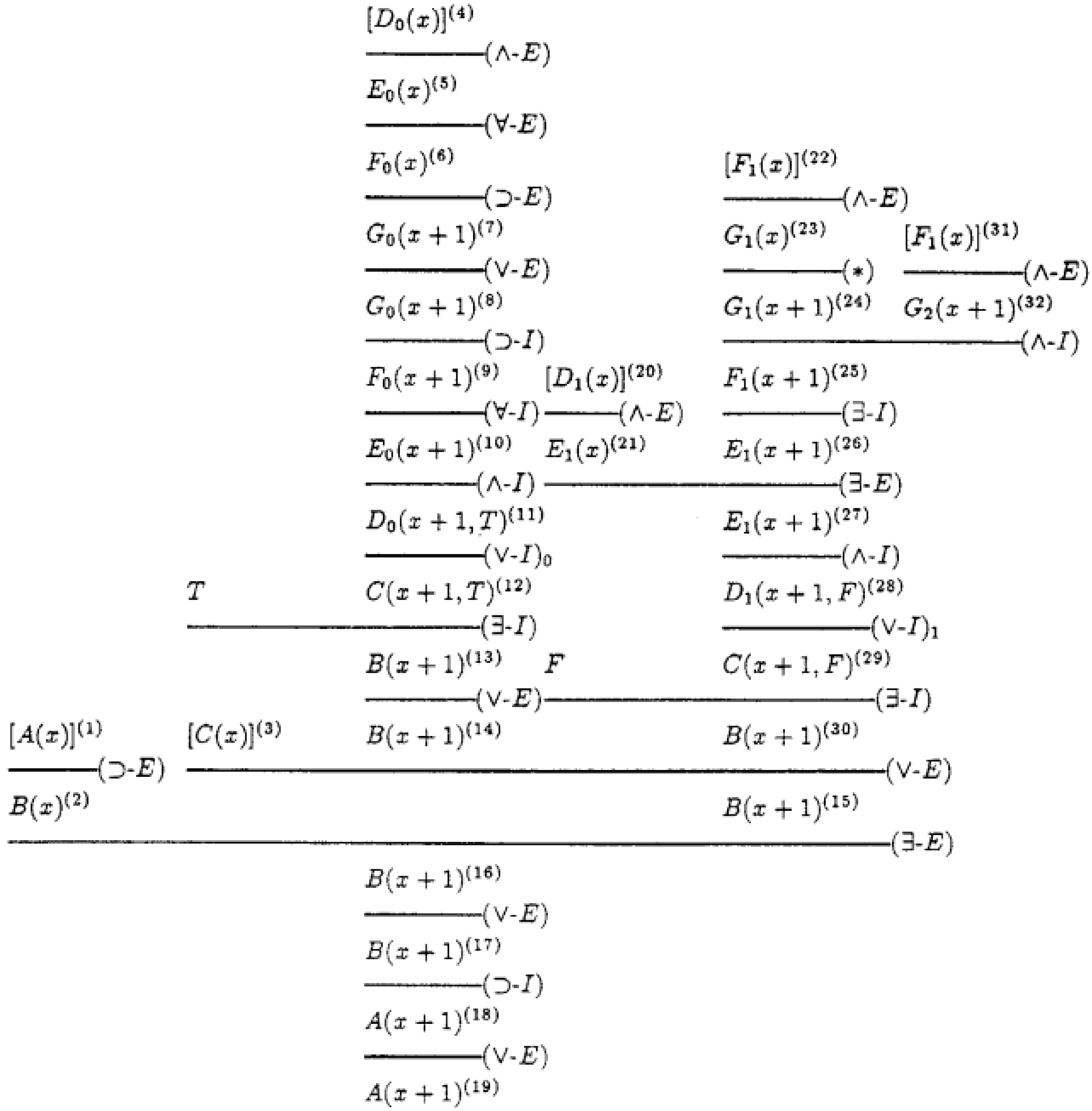

Figure 1

There are four main paths:

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

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

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

and

$S_3 \stackrel{\text{def}}{=} (1),(2),(3),(20),(21),(31),(32),(25),(26),(27),(28),(29),(30),(15),(16),(17),(18),(19)$.

There are six segments: (a) (7), (8); (b) (13), (14), (15), (16), (17); (c) (30), (15), (16), (17); (d) (18), (19); (e) (26), (27); and (f) (32).

Note that this is a normal proof tree, and (a) and (f) are minimum segments, and the sequence (23), (24) is not a segment, but, as stated in section 5, has the same nature as a minimum segment. Segments (b) and (c) are critical.

7.3.2 Initial Marking

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

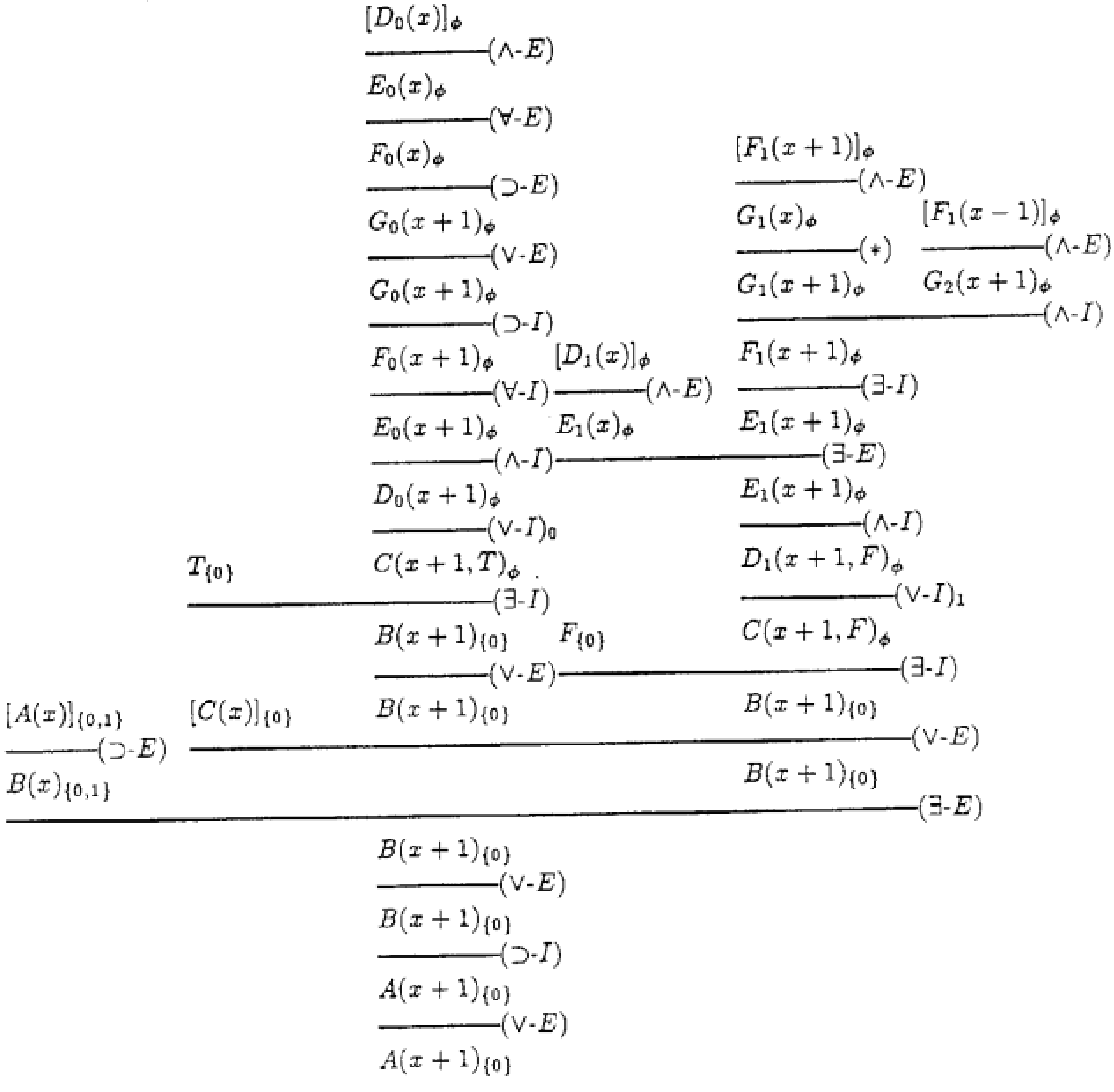

Figure 2

As this proof tree has no critical ($\supset$-E) application but has critical segments, the marking of $A(x)$ (induction hypothesis) is different from the initial declaration of $A(x+1)$ because of the inevitable marking.

7.3.3 Re-marking - Elimination of Critical Segments

Set the declaration to be the same as the marking of $A(x)$ obtained in the previous section, i.e., $\{0,1\}$ and perform the marking again. The obtained marked proof tree is given in Figure 3.

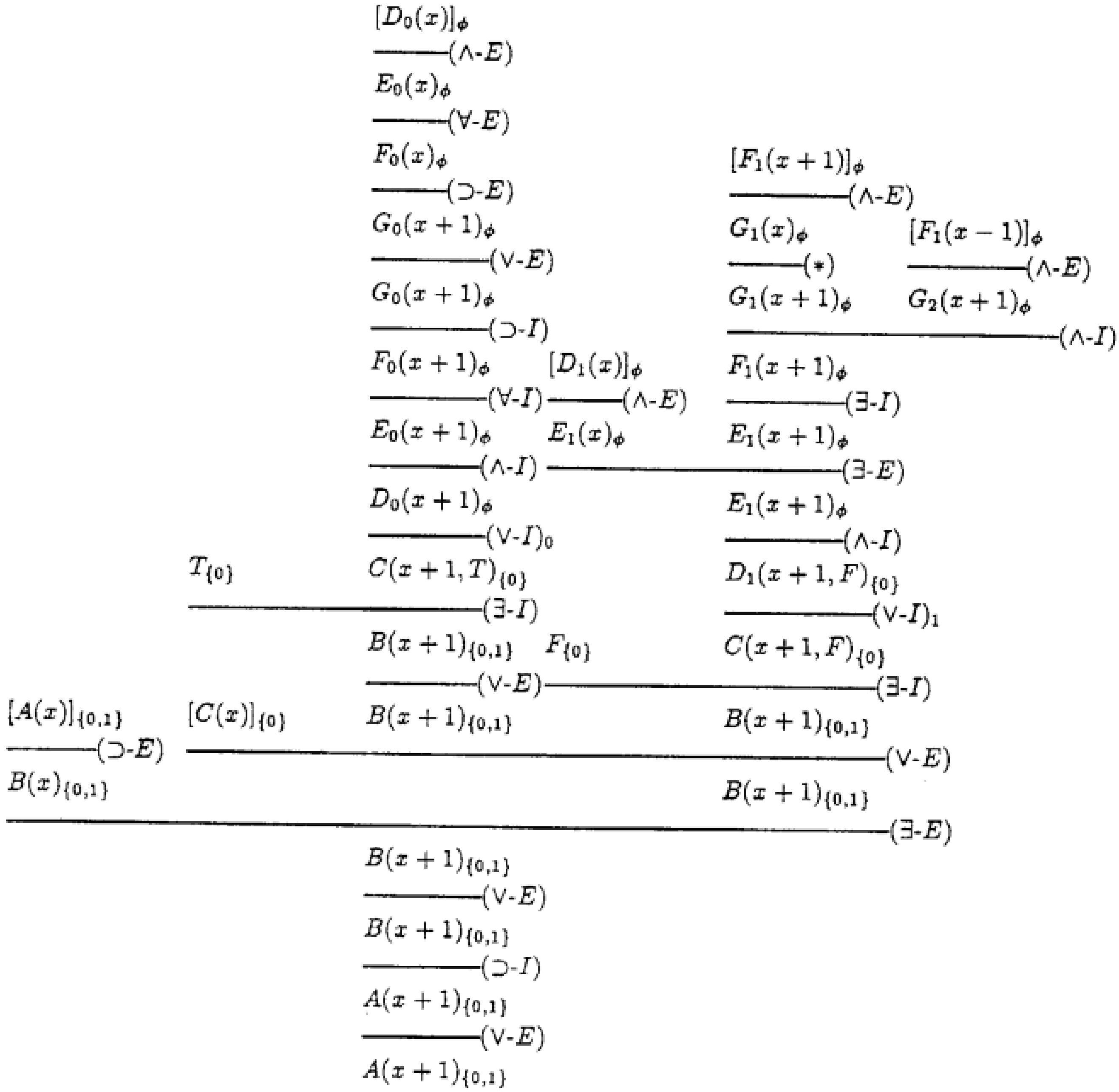

Figure 3

7.3.4 Extraction of Redundancy-free Codes

The code extracted by using the $NExt$ procedure from the marked proof tree obtained in the previous section is as follows:

$$
\begin{aligned}
Ext(\mathbf{lemma'}) \stackrel{\text{def}}{=} \lambda p.\ \mu(z_0, z_1).\ & \\
& \lambda z.\ if\ z = 0 \\
& then\ any[2] \\
& else\ \ if\ z = 1 \\
& \quad then\ any[2] \\
& \quad else\ \ if\ z = 2 \\
& \qquad then\ (T, left) \\
& \qquad else\ \ if\ proj(1)((z_0, z_1)(z-1)) = left \\
& \qquad\quad then\ \ if\ proj(0)(Ext(\mathbf{prop})(p)(z-1)) = left \\
& \qquad\qquad then\ (T, left) \\
& \qquad\qquad else\ (F, right) \\
& \qquad\quad else\ (F, right)
\end{aligned}
$$

Comparing the above code with $Ext(\mathbf{lemma})$, 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, the information whether the input can be divided by a natural number less than the input is necessary, and the information is calculated in the 1th code of the term sequence calculated by the main loop of the multi-valued recursive call function.

8. Conclusion

A proof theoretic method to extract redundancy-free realizer code from a constructive logic was presented in this paper. The realizer codes of standard q-realizability contain some redundancy which can be seen as verification information, and cause heavy runtime overhead. The redundancy can be removed by analysing of the length of formula occurrences in the given proof tree. The crucial part is the analysis of proofs by induction where the ($\vee$-E), ($\exists$-E) and ($\supset$-E) rules are used in particular ways in the proof of induction step. These critical cases are specified from a proof theoretic point of view. The method presented in this paper automatically analyses and eliminates redundancy by making a simple declaration when the theorems and their proofs are set.

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

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

[Hayashi 86] Hayashi, S., "PX: a system extracting programs from proofs", *Proceedings of 3rd Working Conference on the Formal Description of Programming Concepts*, Ebburup, Denmark, North-Holland, 1986

[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

[Huet 86] Huet, G., "*Formal Structure for Computation and Deduction*", Lecture Given at CMU, 1986

[Huet 88] Huet, G., "*A Uniform Approach to Type Theory*", (personal communication)

[Kleene 45] Kleene, S.C., "On the interpretation of intuitionistic number theory", Journal of Symbolic Logic 10, pp109-124, 1945

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

[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 –", *European Symposium on Programming '88*, Nancy, 1988

Appendix Proof of Lemma (Σ)

Main Proof

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

Extracted Code:

$$\begin{array}{l}\lambda p.\ \mu Rv(z \geq 2 \supset A(p,z)).\\ \qquad \lambda z.\ \ if\ z = 0\ then\ Ext(\Sigma_0)\\ \qquad\qquad else\ Ext(p, z, z \geq 2 \supset A(p,z) \vdash (z+1) \geq 2 \supset A(p, z+1))\sigma\end{array}$$

where $\sigma \stackrel{\text{def}}{=} \{Rv(z \geq 2 \supset A(p,z))/Rv(z \geq 2 \supset A(p,z))(z-1)\}$

- Proof of $\vdash 0 \geq 2 \supset A(p,0)$ (Σ_0)

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

Extracted Code:

$$any[l(Rv(A(p,0)))]$$

- Proof of $z,\ p,\ z \geq 2 \supset A(p,z) \vdash (z+1) \geq 2 \supset A(p,z+1)$ (Σ_1)

$$\dfrac{\dfrac{[z : nat]}{z = 0 \vee 1 \leq z}(*) \qquad \dfrac{\dfrac{\dfrac{[z+1 \geq 2] \quad [z=0]}{\perp}}{A(p,z+1)}(\perp\text{-}E)}{z+1 \geq 2 \supset A(p,z+1)} \qquad \dfrac{\begin{array}{c}[z+1 \geq 2]\\ [z \geq 2 \supset A(p,z)]\\ \Sigma_{11}\end{array}}{(z+1) \geq 2 \supset A(p,z+1)}}{(z+1) \geq 2 \supset A(p,z+1)}(\vee\text{-}E)$$

Extracted Code (modified $\vee$-code):

$$if\ z = 0\ then\ any[l(Rv(A(p,z+1)))]\ else\ Ext(z \geq 1, z \geq 2 \supset A(p,z) \vdash z \geq 1 \supset A(p,z+1))$$

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

$$\dfrac{\dfrac{\dfrac{[z \geq 1]}{z = 1 \vee 2 \leq z}(*) \qquad \dfrac{\begin{array}{c}[z=1]\\ \Sigma_{110}\end{array}}{A(p,z+1)} \qquad \dfrac{\begin{array}{c}[z \geq 2]\\ [z \geq 2 \supset A(p,z)]\\ \Sigma_{111}\end{array}}{A(p,z+1)}}{A(p,z+1)}(\vee\text{-}E)}{(z+1) \geq 2 \supset A(p,z+1)}(\supset\text{-}I)$$

Extracted Code (modified $\vee$ code):

$$if\ z = 1\ then\ Ext(z = 1 \vdash A(p,z+1))\ else\ Ext(z \geq 2, z \geq 2 \supset A(p,z) \vdash A(p,z+1))$$

- Proof of Σ_{110}

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

Extracted Code:

$$
T, left, \lambda d.\ any[l(Rv(\neg(d \mid p)))], any[l(P_1(p,2,T))]
$$

- Proof of Σ_{111}

$$
\dfrac{\dfrac{[z \geq 2] \qquad \begin{bmatrix} z \geq 2 \\ \supset A(p,z) \end{bmatrix}}{\exists b.\ P_0(p,z,b) \vee P_1(p,z,b)} \qquad \dfrac{\begin{bmatrix} P_0(p,z,b) \\ \vee P_1(p,z,b) \end{bmatrix} \qquad \dfrac{\begin{matrix}[b] \\ [P_0(p,z,b)] \\ \Sigma_{1110}\end{matrix}}{A(p,z+1)} \qquad \dfrac{\begin{matrix}[b] \\ [P_1(p,z,b)] \\ \Sigma_{1111}\end{matrix}}{A(p,z+1)}}{A(p,z+1)}(\vee\text{-}E)}{A(p,z+1)}(\exists\text{-}E)
$$

Extracted Code:

$$
\begin{pmatrix} if\ proj(1)(Rv(z \geq 2 \supset A(p,z))) = left\ then\ Ext(b, P_0(p,z,b) \vdash A(p,z+1)) \\ \qquad else\ Ext(b, P_1(p,z,b) \vdash A(p,z+1)) \\ \ \end{pmatrix} \sigma
$$

where $\sigma \stackrel{\text{def}}{=} \{b/proj(0)(Rv(z \geq 2 \supset A(p,z)))\}$.

- Proof of Σ_{1110} : $b : nat,\ P_0(p,z,b) \vdash A(p,z) \quad (\stackrel{\text{def}}{=} \exists b.\ P_0(p,z+1,b) \vee P_1(p,z+1,b))$

$$
\dfrac{\dfrac{[z:nat] \qquad \dfrac{[p] \qquad \dfrac{}{\forall m.\ \forall n.\ \neg(n \mid m) \vee (n \mid m)}(\textbf{Prop.})}{\forall n.\ \neg(n \mid p) \vee (n \mid p)}}{\neg(z \mid p) \vee (z \mid p)} \qquad \dfrac{\begin{matrix}[P_0(p,z,b)] \\ [\neg(z \mid p)] \\ \Sigma_{11100}\end{matrix}}{A(p,z+1)} \qquad \dfrac{\begin{matrix}[P_1(p,z,b)] \\ [(z \mid p)] \\ \Sigma_{11101}\end{matrix}}{A(p,z+1)}}{\exists b.\ P_0(p,z+1,b) \vee P_1(p,z+1,b)}(\vee\text{-}E)
$$

Extracted Code:

$$
\begin{aligned} &if\ proj(0)(Ext(\textbf{Prop.})(p)(z)) = left\ then\ Ext(P_0(p,z,b), \neg(z \mid p) \vdash A(p,z+1)) \\ &\qquad\qquad else\ Ext(P_0(p,z,b), (z \mid p) \vdash A(p,z+1) \end{aligned}
$$

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

$$
\dfrac{\overline{T} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[1<d<z+1]}{1<d<z \vee d=z} \qquad \dfrac{\begin{bmatrix}1<d\\<z\end{bmatrix} \qquad \dfrac{[d] \qquad \dfrac{[P_0(p,z,b)]}{\forall d.\ 1<d<z \supset \neg(d \mid p)}(\wedge\text{-}E)}{1<d<z \supset \neg(d \mid p)}}{\neg(d \mid p)} \qquad \dfrac{[d=z] \qquad [\neg(z \mid p)]}{\neg(d \mid p)}}{\neg(d \mid p)}}{1<d<(z+1) \supset \neg(d \mid p)}}{\forall d.\ 1<d<(z+1) \supset \neg(d \mid p)} \qquad \dfrac{\overline{T}^{(*)}}{T=T}}{P_0(p,z+1,T)}}{P_0(p,z+1,T) \vee P_1(p,z+1,T)}}{\exists b.\ P_0(p,z+1,b) \vee P_1(p,z+1,b)}
$$

Extracted Code (modified $\vee$ code):

$$T, left, \lambda d.\ if\ 1<d<z\ then\ nil\ else\ nil, any[l(P_1(p,z,T))]$$

- Proof of Σ_{11101} : $b : nat,\ \neg(z \mid p),\ P_0(p,z,b) \vdash A(p, z+1)$

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

Extracted Code:

$$F, right, any[l(P_0(p,z+1,F))], z, Rv((z \mid p))$$

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

$$
\dfrac{\overline{F}^{(*)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{[P_1(p,z,b)]}{\exists d.\ 1<d<z \wedge (d \mid p)} \qquad \dfrac{[d:nat] \qquad \dfrac{\dfrac{\dfrac{\begin{bmatrix}1<d\\<z\\\wedge(d \mid p)\end{bmatrix}}{1<d<z} \qquad \dfrac{[z:nat]}{z<z+1}}{1<d<z+1} \qquad \dfrac{\begin{bmatrix}1<d\\<z\\\wedge(d \mid p)\end{bmatrix}}{(d \mid p)}}{1<d<z+1 \wedge (d \mid p)}}{\exists d.\ 1<d<z+1 \wedge (d \mid p)}}{\exists d.\ 1<d<z+1 \wedge (d \mid p)}(\exists\text{-}E) \qquad \dfrac{\overline{F}}{F=F}}{P_1(p,z+1,F)}}{P_0(p,z+1,F) \vee P_1(p,z+1,F)}}{\exists b.\ P_0(p,z+1,b) \vee P_1(p,z+1,b)}
$$

Extracted Code:

$$F, right, any[l(P_0(p,z+1,F))], (d, Rv((d \mid p)))\sigma$$

where

$\sigma \stackrel{\text{def}}{=} \{d/proj(0)Rv(P_1(p,z,b)),$
$\quad Rv((d \mid p))/tseq(1)Rv(P_1(p,z,b))\}.$

Proposition

$\forall m : nat.\ \forall n : nat.\ \neg(n \mid m) \vee (n \mid m)$

The code extracted from the proof of this proposition is to divide m by n to calculate the quotient and the remainder, and returns the quotient if the remainder is zero other wise returns any code.

Proof of Proposition

$$\dfrac{\dfrac{\dfrac{[n : nat] \quad \dfrac{[m : nat] \quad \dfrac{}{\forall p.\ \forall q.\ \exists d.\ \exists r.\ (p = d \cdot q + r \wedge 0 \leq r < q)}(\mathbf{Th.})}{\forall q.\ \exists d.\ \exists r.\ (m = d \cdot q + r \wedge 0 \leq r < q)}}{\exists d.\ \exists r.\ (m = d \cdot n + r \wedge 0 \leq r < n)} \quad \Pi}{\neg(n \mid m) \vee (n \mid m)}(\exists\text{-}E)}{\dfrac{\forall n.\ \neg(n \mid m) \vee (n \mid m)}{\forall m.\ \forall n.\ \neg(n \mid m) \vee (n \mid m)}}$$

Extracted Code:

$$\lambda m.\ \lambda n.\ (if\ 0 = r\ then\ right, d\ else\ left, any[Rv((n \mid m))])\sigma$$

where $\sigma \stackrel{\text{def}}{=} \{d/proj(0)(Ext(\mathbf{Th.})),\ r/proj(1)(Ext(\mathbf{Th.}))\}$. **Th.** is the theorem of natural number division.

Π

$$\dfrac{[\exists r.\ (m = d \cdot n + r \wedge 0 \leq r < n)] \quad \dfrac{\dfrac{\dfrac{[m = d \cdot n + r \wedge 0 \leq r < n]}{0 \leq r < n}}{0 = r \vee 0 < r < n} \quad \begin{matrix}[0 = r] \\ \Pi_a\end{matrix} \quad \begin{matrix}[0 < r < n] \\ \Pi_b\end{matrix}}{\neg(n \mid m) \vee (n \mid m)}}{\neg(n \mid m) \vee (n \mid m)}(\exists\text{-}E)$$

Π_a

$$\dfrac{\dfrac{[d] \quad \dfrac{[0 = r] \quad \dfrac{[m = d \cdot n + r \wedge 0 \leq r < n]}{m = d \cdot n + r}}{m = d \cdot n}}{(n \mid m)}}{\neg(n \mid m) \vee (n \mid m)}$$

Π_b

$$\dfrac{\dfrac{\dfrac{[(n \mid m)] \quad \dfrac{\dfrac{[0 < r < n] \quad \Pi_c}{(d' - d) : nat}(*) \quad \dfrac{[0 < r < n] \quad \Pi_c}{0 < (d' - d) < 1}(*)}{\bot}}{\bot}}{\neg(n \mid m)}(\supset\text{-}I)}{\neg(n \mid m) \vee (n \mid m)}$$

Proof of Π_c:

$$\dfrac{\dfrac{[m = d \cdot n + r \wedge 0 \leq r < n]}{m = d \cdot n + r} \quad [m = d' \cdot n]}{(d' - d) \cdot n = r}$$