ソフトウェア工学 2-3
(1977. 7. 20)

An Algorithm of Extracting a Program from a Proof of its Specification

T. Kamimura

Toshiba Research and Development Center
1 Komukai Toshiba-cho, Saiwai-ku, Kawasaki-shi 210, Japan

Abstract

A program extraction algorithm is presented, which, given a formal proof of a formula in a logical system, generates a flow-chart program having this formula as its specification. Natural Deduction system is adopted as the logical system in which actual processes of proving a formula are carried out. This is because the system goes well along with our natural reasoning in proving a formula and therefore can be easily enhanced by the introduction of interactions with humans. Thus, the method provides a sufficient basis for constructing a correct program by proving its specification in a conversational theorem proving system.

1. Introduction

Program synthesis is the construction of a computer program from given specifications. Approaches taken so far are classified into three groupes: formal approaches, heuristics and programming methodology such as structured programming. In these approaches, the first ones have been studied most.[1)2)] Their proposed systems are well specified and some of them are actually implemented. These systems have been proven to be sufficient for a simple problem. These approaches are aiming at fully automating a process of synthesizing a program, but it seems difficult to automate the construction of a complex program. An additional difficulty is that the systems are not easily enhanced by the introduction of interactions with humans. This is because the program synthesis processes in the systems are not along with our natural reasoning in constructing a program.

In this report, we present a new program extraction method which will go along with our natural reasoning and easily incorporate interactions with humans. Since our present concern is to provide a fundamental algorithm, the domain of a program will be restricted to the area of natural numbers. Taking a constructive property such as shown in 3) into account, we adopt HA (Heyting Arithmetic) as a basic formalism of our method, giving it in terms of Natural Deduction style, a fomalism by G. Gentzen. Based on the formalism, we give an extraction algorithm which, given a formal proof of a formula in HA, generates a flow-chart program having this formula as its specification. Thus, the method provides a sufficient basis for constructing a correct program by proving its specification in a conversational theorem proving system.

2. HA

A formal system HA is a first order theory which was formulated to represent an intuitionistic natural number theory. The system HA is characterized as a formulation of our constructive analysis in the area of natural numbers, and it is known that HA is related to the recursive functions.[3)4)] On the other hand, as our approach aims at introducing the interactions with humans into theorem proving system, Natural Deduction style seems appropriate to our system. Thus, we give HA in terms of NJ shemata[5)]. In the following, an ordinary first order language for natural numbers is assumed. Variables, terms, formulas, wffs, atomic formulas and so forth are understood as usual meanings.

Axioms: If an atomic formula A becomes true when it is interpreted intuitionistically, A is an axiom. Nothing else is an axiom.

Inference Rule:

1) &-I $\quad \dfrac{A \quad B}{A \,\&\, B}$ $\qquad$ 2) &-E $\quad \dfrac{A \,\&\, B}{A}$ $\quad \dfrac{A \,\&\, B}{B}$ $\qquad$ 3) ∨-I $\quad \dfrac{A}{A \vee B}$ $\quad \dfrac{B}{A \vee B}$

4) ∨-E $\quad \dfrac{A \vee B \quad \overset{[A]}{C} \quad \overset{[B]}{C}}{C}$ $\qquad$ 5) ∀-I $\quad \dfrac{F(a)}{\forall x F(x)}$ $\qquad$ 6) ∀-E $\quad \dfrac{\forall x F(x)}{F(t)}$

7) ∃-I $\quad \dfrac{F(t)}{\exists x F(x)}$ $\qquad$ 8) ∃-E $\quad \dfrac{\exists x F(x) \quad \overset{[F(a)]}{A}}{A}$ $\qquad$ 9) ⊃-I $\quad \dfrac{\overset{[A]}{B}}{A \supset B}$

10) ⊃-E $\quad \dfrac{A \quad A \supset B}{B}$ $\qquad$ 11) MJ $\quad \dfrac{F(0) \quad \forall x(F(x) \supset F(x'))}{\forall x F(x)}$

where x' is a successor of x.

12) $\sim$-I_1 $\quad \dfrac{\overset{[A]}{B}}{\sim A}$ $\qquad$ 13) $\sim$-I_2 $\quad \dfrac{\overset{[A]}{B} \quad \overset{[A]}{\sim B}}{\sim A}$ $\qquad$ 14) $\sim$-E $\quad \dfrac{A \quad \sim A}{B}$

where B is intuitionistically false.

where B is any atomic formula.

The free object variable of ∀-I or ∃-E, designated by a in the respective schema is an eigenvariable. A symbol t of ∀-E or ∃-I denotes any term which is free for x in F(x). Use of ⊃-I is restricted so that only one occurrence of A is removed at a time from assumptions of B. Therefore, if there are n assumptions of A in proving B, the schema gives $\underbrace{A \supset A \supset \cdots \supset A}_{\text{n times}} \supset B$ instead of $A \supset B$. Functions and predicates appearing in HA are assumed to be computable.

3. Programs

In this section, a program is specified in terms of a flow-chart language. Besides the usual elements of flow-chart figures, we adopt two additional mechanisms in specifying a program. The one is to admit a procedure definition including a recursive call mechanism. The other is to use a program variable taking a program or a procedure itself as its value. This mechanism may be similar to the function in ALGOL of calling a procedure with another procedure as its actual parameter. In addition to the program variable, two other variables are used to give a program; namely an individual variable taking a natural number as its value, and a logical variable taking true or false as its value.

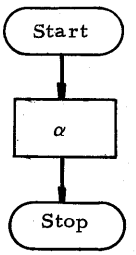
Definition 3.1 A program p is specified by:
1) An input variable list i(p) and an output variable list o(p). Each element of these lists is an individual, logical or program variable. Values of the variables of i(p) cannot be changed in p.
2) A flow-chart figure obtained by connecting a number of boxes represented in Fig. 1 as usual so that it starts from the box 1) and ends in the box 2).

Similarly, a procedure is specified by a flow-chart figure obtained by connecting a number of boxes in Fig. 1 as usual so that it starts from the box 3) and ends in the box 4). Here, p_{name} is a procedure name, and $(x_1 \cdots x_n)$ and $(y_1 \cdots y_m)$ are input and output variable lists, respectively. Values of $x_1 \cdots x_n$ cannot be changed in this procedure.

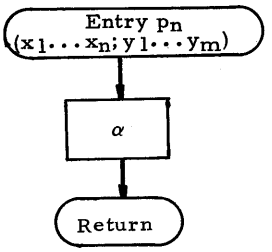
It is assumed that there are infinite number of individual, logical and program variables.

From this definition, a following lemma is straightforward.

Lemma 3.1 If a program p is



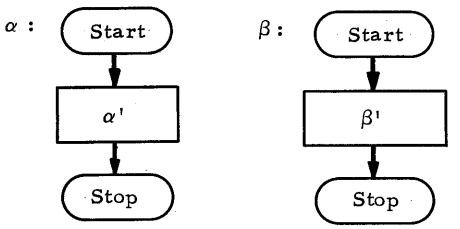
then



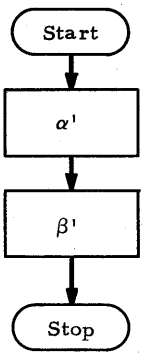
is a procedure, where i(p) is $(x_1 \cdots x_n)$ and o(p) is $(y_1 \cdots y_m)$ respectively.

As a matter of fact, when a program vabiable is to take a program as its value, it is realized by assigning a name of a new procedure which is introduced from the program by this lemma, to the variable. Thus, a value of a program variable is always given in the form of a procedure name.

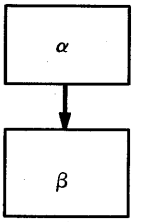
In the following, we use an abbrivation in combining a couple of programs to make up another program. Namely, if a program α and β are



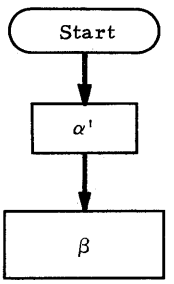
we abbriviate



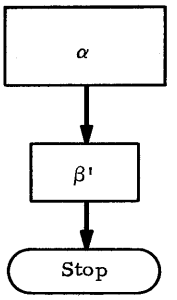
to



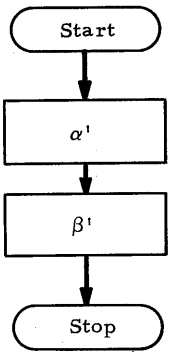
, while if α' or β' is one of the boxes in Fig. 1

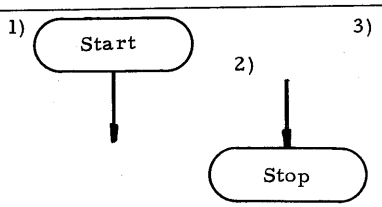


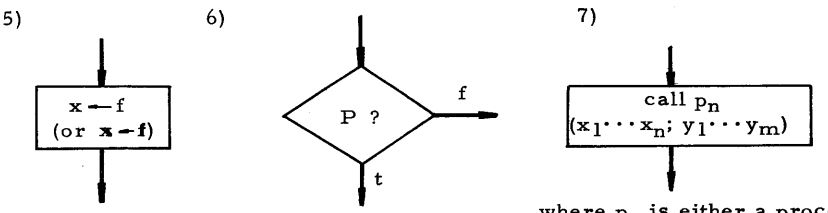
or



means







where P is a computable predicate.

where p_n is either a procedure name or a program variable.

Fig. 1 Flow-chart language

according as α' or β' is one of the boxes.

In general, procedures can be taken as special forms of programs in describing their properties. Therefore, programs are, hereafter, considered to include procedures unless specified to the contrary.

4. Realization of a wff

Next, we give a relation between a program and a wff in HA. Hereafter, it is assumed that each bound variable appearing in a waff is distinct.

Definition 4.1 (Type)A type is defined as follows.

1) Symbols ω and ρ denote basic types.

2) A symbol ϕ_τ is a listed type.

3) If t_i $(1 \leq i \leq n)$ is a basic type or a compound type, a list $(t_1 \cdots t_n)$ is a listed type.

4) If t_d and t_r are listed types, $t_d \rightarrow t_r$ is a compound type.

5) Nothing else is a type.

Definition 4.2

1) Any natural number or any individual vaviable has the type ω.

2) Any logical variable, true or false has the type ρ.

3) A null list has the type ϕ_τ.

4) A list $(u_1 \cdots u_n)$ has the type $(t_1 \cdots t_n)$ if each u_i $(1 \leq i \leq n)$ has the type t_i.

5) A program p such that i(p) and o(p) have the types t_d and t_r respectively, or a program variable which is supposed to take a procedure whose input and output lists have the types t_d and t_r respectively, as its value has the type $t_d \rightarrow t_r$.

When objects α and β have the same types, we denote this as $\alpha \underset{t}{=} \beta$.

Definition 4.3 Let A be a wff in HA. Then, input list I(A) and output list O(A) are defined inductively as follows. Hereafter, bound variables appearing in a wff are considered to be also individual variables.

1) A : atomic formula

$I(A) = \phi$

$O(A) = \phi$

2) $I(A \vee B) = Conc(I(A), I(B))$

$O(A \vee B) = Conc(O(A), O(B))$

3) $I(A \,\&\, B) = Conc(I(A), O(B))$

$O(A \,\&\, B) = Conc(O(A), O(B))$

4) $I(\forall x F(x)) = Conc((x), I(F(x)))$

$O(\forall x F(x)) = O(F(x))$

5) $I(\forall x F(x)) = I(F(x))$

$O(\forall x F(x)) = Conc((x), O(F(x)))$

6) $I(A \supset B) = Conc((v_p), I(B))$

$O(A \supset B) = Conc(I(A), O(B))$

7) $I(\sim A) = (v_p')$

$O(\sim A) = I(A)$,

where ϕ indicates a null list, and v_p or v_p' is a program variable. A type of v_p or v_p' is $t_d \rightarrow t_r$, where t_d or t_r is a type of I(A) or O(A), respectively. A symbol Conc means a function of two arguments which concatenates its second argument after its first argument, such as

$$Conc((u_1 \cdots u_n)\ (u_1' \cdots u_m')) = (u_1 \cdots u_n\ u_1' \cdots u_m')$$

Definition 4.4 Let A be a wff in HA, and p a program. Then, p meets A if and only if $i(p) \underset{t}{=} I(A)$ and $o(p) \underset{t}{=} O(A)$.

Definition 4.5 Let A be a wff in HA, and ℓ_1 and ℓ_2 lists whose each element is a natural number or a program. Suppose $\ell_1 \underset{t}{=} I(A)$ and $\ell_2 \underset{t}{=} O(A)$, a program $Ev[A(\ell_1, \ell_2)]$ is defined as follows.

An input variable list of the program is ϕ, and an output variable list is (v_t), where v_t is a logical variable.

1) A : atomic formula. In this case, ℓ_1 and ℓ_2 must be null lists. Then, $Ev[A(\ell_1, \ell_2)]$ is a program of assigning true or false to v_t as its value according to the result of evaluating a formula A. If there are free variables in A, they are considered to have definite values. This program can be always

constructed because A is quantifier free and does not contain non-computable functions or predicates.

2) A = BVC Then $[A(\ell_1, \ell_2)]$ is given as follows.

Fig. 2-1

3) A = B & C

Fig. 2-2

4) A = $\forall x F(x)$. In this case, $I(\forall x F(x))$ is $Conc((x), I(F(x)))$. Therefore, ℓ_1 must be of the form $Conc((n), \ell_1')$, where n is a natural number. A program $Ev[\forall x F(x)(\ell_1, \ell_2)]$ is

Fig. 2-3

5) A = $\exists x F(x)$. In this case, $O(\exists x F(x))$ is $Conc((x), F(x))$. Therefore, ℓ_2 must be of the form $Conc((n), \ell_2')$, where n is a natural number. A program $Ev[\exists x F(x)(\ell_1, \ell_2)]$ is

Fig. 2-4

6) A = B ⊃ C. In this case, $I(B \supset C)$ is $Conc((v_p), I(C))$. Therefore, ℓ_1 must be of the form $Conc((q), \ell_1')$, where q is a program, and ℓ_2 must be of the form $Conc(\ell_2', \ell_2'')$. $Ev[B \supset C(\ell_1, \ell_2)]$ is

Fig. 2-5

7) A = ~B. Then, ℓ_1 must be (q), where q is a program. A program $Ev[\sim B(\ell_1, \ell_2)]$ is given as follows.

Fig. 2-6

Definition 4.6 Let p be a program, and ℓ a list $(u_1 \cdots u_n)$, where each $u_i (1 \leq i \leq n)$ is a natural number or a program. Suppose a list of first n elements of i(p) and ℓ have the same types, $p(\ell)$ indicates a program obtained from p after taking ℓ as its first n inputs. Therefore, if i(p) is $(x_1 \cdots x_n\ x_{n+1} \cdots x_m)$, $i(p(\ell))$ is $(x_{n+1} \cdots x_m)$. When $i(p(\ell))$ is ϕ, in other words, i(p) and ℓ have the same types, $\overline{p(\ell)}$ is a list of output values of $p(\ell)$.

Definition 4.7 Suppose a program p meets a wff A, then p is said to realize A if and only if $\overline{Ev[A(\ell, \overline{p(\ell)})]}$ is always (true) for any list ℓ such that $\ell \underset{t}{=} I(A)$.

Definition 4.8 Let A be a wff in HA. A program Ev{A} in another program p is a program $Ev[A(\ell_1, \ell_2)]$ in which ℓ_1 and ℓ_2 are lists of values of the variables of I(A) and O(A) at the point in the program p. If some of these values are not given, each of them is defined as 0, true or $\phi_p^{(*)}$, according as the variable is individual, logical or program variable.

5. Extraction of Programs

Now, we are ready to give a program extraction method. This is done by proving a following theorem.

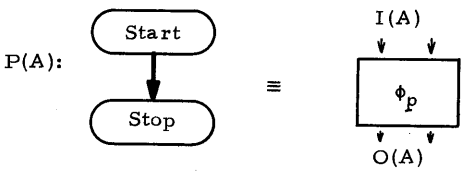
Theorem If a wff A is provable in HA, we can construct a program p which realizes A.

The proof is by induction on the length of the given deduction of A. At each step, an extraction operator P is defined which actually generates a desirable program.

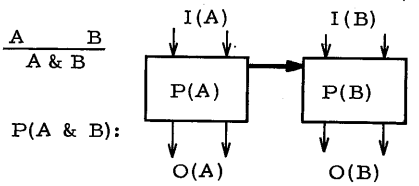
Proof) In the following, it is assumed that ⇢ indicates a relation of variables' references and ➡ indicates a flow of a control of a program.

First, i(p) and o(p) are given as I(A) and O(A), respectively.

1) A : axiom

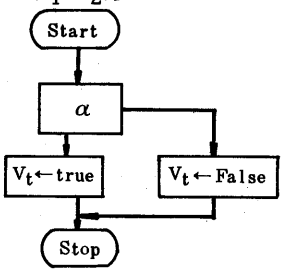


2) $\frac{A \qquad B}{A \,\&\, B}$



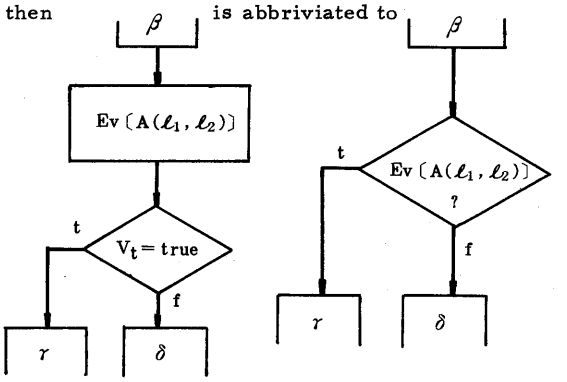
(*) See 1) in next section.

Hereafter, if Ev[A(ℓ_1 , ℓ_2)] is



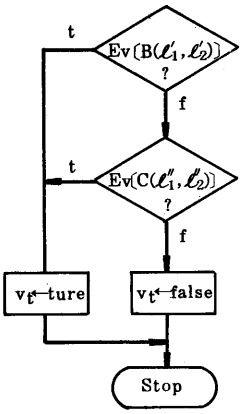
,

then

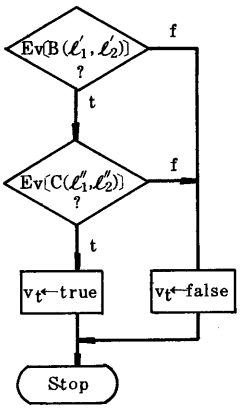


is abbriviated to

1)

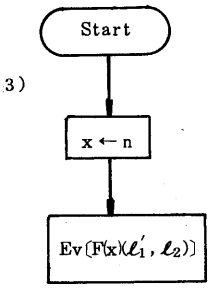


2)

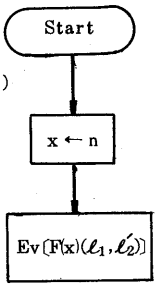


In 1) and 2), ℓ_1 and ℓ_2 are considered to be Conc(ℓ_1', ℓ_1'') and Conc(ℓ_2', ℓ_2'') respectively, where $\ell_1' \underset{t}{=} I(B)$, $\ell_2' \underset{t}{=} O(B)$, $\ell_1'' \underset{t}{=} I(C)$, and $\ell_2'' \underset{t}{=} O(C)$.

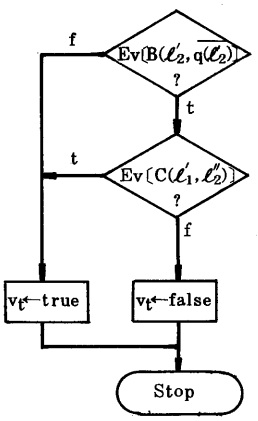
3)



4)



5)



6)

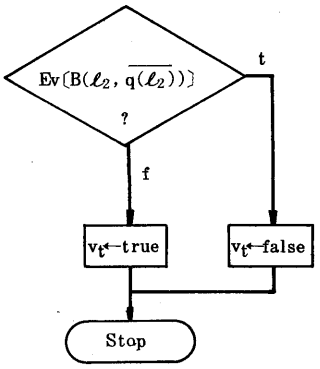
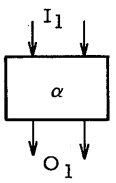


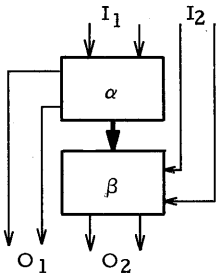
Fig. 2

3) $\frac{A \,\&\, B}{A}$

P(A):



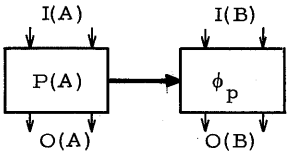
, where P(A & B) is



4) $\frac{A \,\&\, B}{B}$ Similarly to 3).

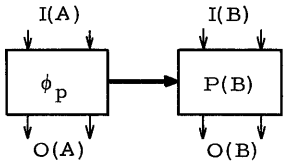
5) $\frac{A}{A \vee B}$

P(AVB):



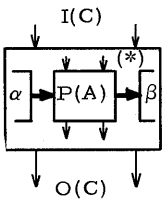
6) $\frac{B}{A \vee B}$

P(AVB):

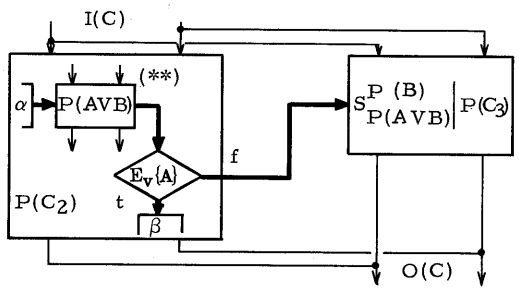


7) $\frac{[A]\ [B]}{A \vee B \quad C_2 \quad C_3}{C_1}$

When P(C_2) is given as



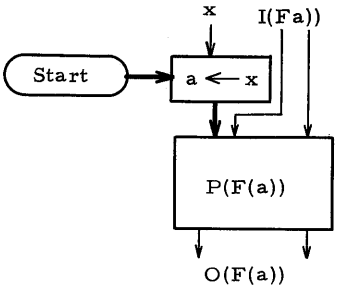
P(C_1) is defined as follows.



where $S^{\gamma}_{\delta}|E$ is a program obtained by substituting δ for every γ in a program E.

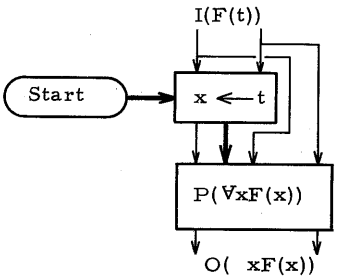
8) $\frac{F(a)}{\forall x F(x)}$

P(∀xF(x)):



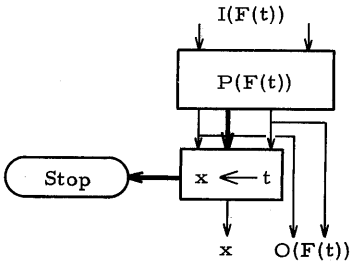
9) $\frac{\forall x F(x)}{F(t)}$

P(F(t)):



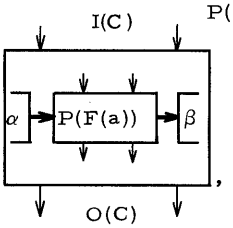
10) $\frac{F(t)}{\exists x F(x)}$

P(xF(x)):



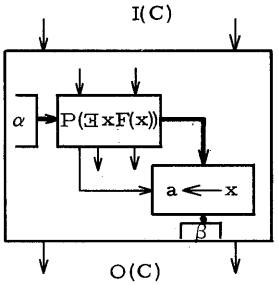
11) $\frac{[F(a)]}{\exists x F(x) \quad C_2}{C_1}$

When P(C_2) is given as



,

P(C_1):



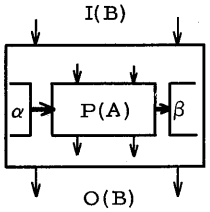
(*) If P(A) is a procedure, P(A) in P(C) means the box where the procedure P(A) is being called.

(**) If the values of I(B) are not defined here, they are given similarly in Definition 4.8.

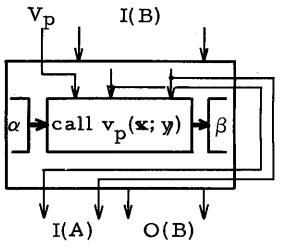
The two rules of (*) and (**) apply to similar cases appearing in the subsequent part of this paper.

12) $\dfrac{[A]}{\dfrac{B}{A \supset B}}$

If P(B) is



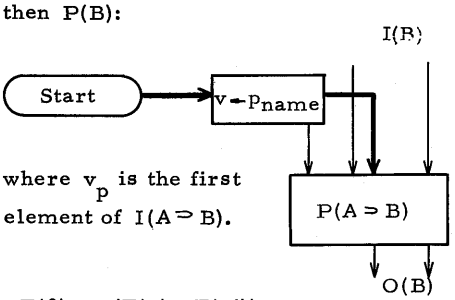
P(A ⊃ B):



where x = I(A) and y = O(A).

13) $\dfrac{A \quad A \supset B}{B}$

First, a procedure p_{name} is defined from P(A) by the lemma 3.1, then P(B):



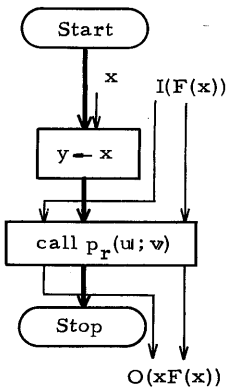
where v_p is the first element of $I(A \supset B)$.

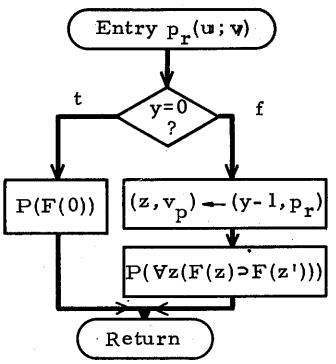
14) $\dfrac{F(0) \quad \forall x(F(x) \supset F(x'))}{\forall x F(x)}$

where x' is a successor of x.

A new procedure p_r is defined as follows.

P(∀xF(x)):



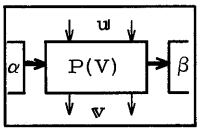


where I(∀yF(y)) and O(∀yF(y)) are u and v, respectively, and the first element of u must be y. A variable v_p is the first element of $I(F(z) \supset F(z'))$.

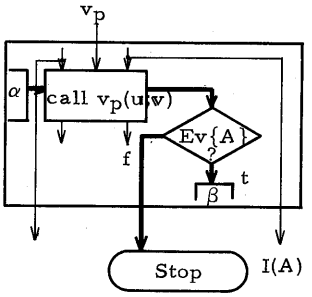
15) $\dfrac{[A]}{\dfrac{B}{\sim A}}$

where B is false when it is evaluated intuitionistically.

When P(B) is



then P(~A):

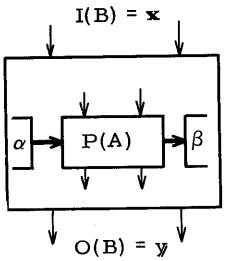


where I(A) is u and O(A) is v.

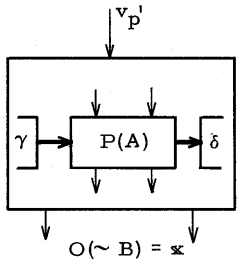
16) $\dfrac{[A] \quad [A]}{\dfrac{B \quad \sim B}{\sim A}}$

By the hypothesis of the induction, P(B) and P(~B) are given as follows.

P(B):

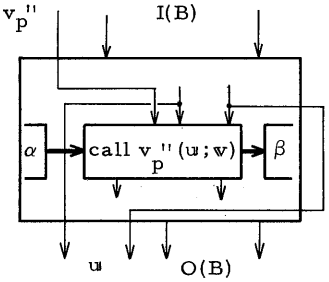


P(~B):

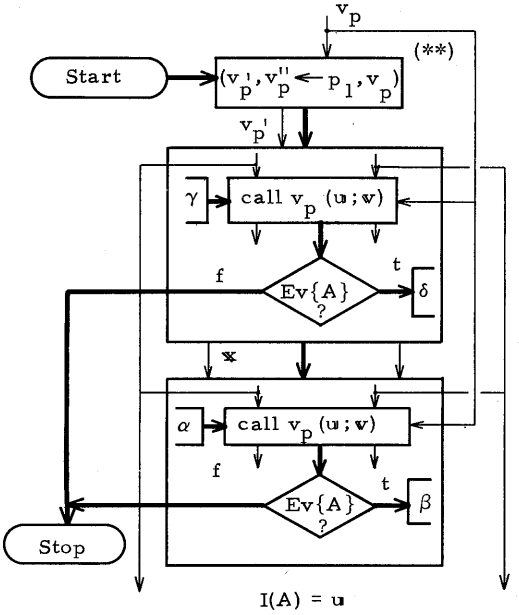


Let I(~A) be (v_p). First a procedure p_1 is defined by the lemma 3. 1 from the following program P(B)" (= P(A ⊃ B)).

(*)
P(B)"



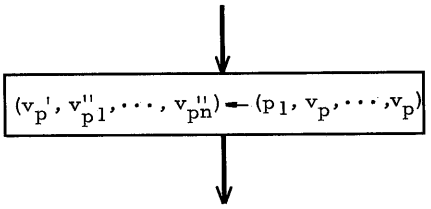
Then, P(~A) is



where w is O(A).

(*)(**) In 16), it is assumed that there is only one occurrence of P(A) in P(B). If there are n(n > 1) occurrences of P(A) in P(B), a program P(~A) is modified as follows.

The program P(B") is obtained in the same way as the case of constructing $P(\underbrace{A \supset A \supset \cdots \supset A}_{n \text{ times}} \supset B)$ from given P(B), assuming n times of P(A) is included in P(B). Therefore, i(P(B")) and O(P(B")) are $Conc(v''_{p1} \cdots v''_{pn})$, I(B)) and $Conc(u_1 \cdots u_n)$, O(B)), respectively. Furthermore, an assignment of (**) is replaced by:



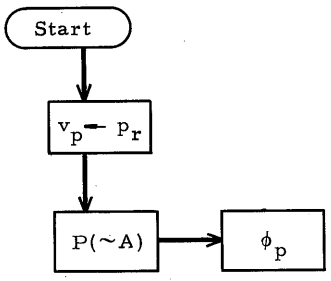
17) $\dfrac{A \quad \sim A}{B}$

where B is any atomic formula.

First a procedure p_r is defined from a program P(A) by the lemma 3. 1.

Then,

P(B):



This means that ϕ_p can be considered as a realization of B by means of the logical deductions which were used to prove the realizations of P(A) and P(~A).

6. Correctness of P

Concerning the cases from 1) to 15), their correctness is obvious. We, then, explain the correctness of 16).

Assuming Ev{A} in P(~B) always out-

puts (true), $\mathfrak{x}$ gives a counter example to the realization of a program $p_1(v_p)$ by the hypothesis of the induction on P(~B). Therefore, $\overline{Ev[B(\mathfrak{x}, \overline{p_1(v_p)(\mathfrak{x})})]}$ is (false). In addition, if $Ev\{A\}$ in P(B) also outputs (true), then a program P(B) and a procedure $p_1(v_p)$ become to be exactly the same. By the hypothesis of the induction on P(B), $\overline{Ev[B(\mathfrak{x}, \mathfrak{y})]}$ is (true). Since $\mathfrak{y}$ is the same as $p_1(v_p)(\mathfrak{x})$, this leads to a contradiction. In other words, at least one of $Ev\{A\}$'s in P(~B) or P(B) must give (false) and then, it transfers a control to the box "Stop". Thus, $\mathfrak{u}$ always gives a counter example to the realization of A.

References

1) C. Green., "Theorem Proving by Resolution as a basis for Question-Answering Systems" in Machine Intelligence. vol.4, pp.183-205, American Elsevier, New York, 1969

2) R. C. T. Lee, C. L. Chang and R. J. Waldinger., "An Improved Program-Synthesizing Algorithm and Its Correctness" C. ACM, vol.17, No.4, pp.211-217, 1974

3) S. C. Kleene., Introduction to Metamathematics, North-Holland, 1967

4) J. R. Shoenfield., Mathematical Logic, Addison-Wesley, 1967

5) G. Gentzen., "Untersuchungen über das logistische Schliessen I, II" Math. Z. 39, pp.176-210, pp.405-431, 1935

6) K. Gödel., "Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes" Dialectica. 12, pp.76-82, 1958