Publ. RIMS, Kyoto Univ.
19 (1983), 237-261

A Proof Description Language and Its Reduction System

By

Masami HAGIYA*

Abstract

The normalization of a natural deduction proof of a closed existential formula $\exists xA$ gives a term t and a proof of $A_x[t]$. This allows us to regard a proof as a program (Goad [9] [10] etc.). But it is not always necessary to completely normalize the given proof to obtain t. We analyze the situation by introducing the notions called *minimal I-reduct, proper reduction* etc.; in a word, we define the *normal order* of proof reduction and study its proof-theoretical property. Then, we present an experimental proof-checker-reducer system that actually uses those principles. In designing a proof-checker (or rather a *proof description language*), we focussed our attention on the readability of proofs.

§ 1. Introduction

A theory in which if a closed existential formula $\exists xA$ is provable, then $A_x[t]$ is also provable for some closed term t is said to have ED (explicit definability property). So called constructive theories have ED, while classical ones do not have ED in general; e.g. Peano's arithmetic (PA) does not have ED (cf. Gödel's Incompleteness Theorem).

We take Heyting's arithmetic (HA) as a typical case of constructive theories. There are several methods of extracting the term t from the proof of $\exists xA$:

- ❶ recursive realizability (Kleene [15] [16], Nelson [21])
- ❷ modified realizability (Kreisel [17])
- ❸ normalization (Prawitz [22] [23])
- ❹ cut elimination (Gentzen [8])
- ❺ Dialectica interpretation (Gödel [11]).

The stability of the E-theorems (i.e. the above methods all compute the same t for $\exists xA$) is discussed in Mints [20], Diller [6].

Among the methods above, Prawitz' normalization of the natural deduction is the most fundamental and the simplest; it is also easily implementable on a

Received April 12, 1982.
* Research Institute for Mathematical Sciences, Kyoto University, Kyoto 606, Japan.

computer. But its serious drawback lies in its inefficiency. The major reasons for this are:

(i) The data structure for representing proof trees is not appropriate for proof reduction.
(ii) It is not always necessary to normalize the given proof; fewer reduction steps are often enough to compute the answer (i.e. the term t).
(iii) The repertoire of induction rules is poor.

Goad [9] [10] is a comprehensive study of the subject, where he solved (i) by introducing what he called a p-term, (ii) by allowing to use an arbitrary Harrop formula as an axiom and (iii) by introducing the so-called recursive proofs.

In this paper, we study the problem (ii) above, especially relating it with the lambda calculus. Why this problem is important is that, as Goad argued, the most part of a proof is irrelevant for the computation i.e. for finding the answer. Then a question arises as to whether or not the irrelevant part can be automatically eliminated or left untouched when executing (or reducing) the proof. This paper answers the question positively.

In Section 2 of the paper, we reformulate **NM** of Prawitz. The formulation is affected by the formulae-as-types notion (de Brujin [3], Howard [12], Diller [6], Martin-Löf [19], Scott [24], de Vrijer [26]).

In Section 3, we introduce the notions called *minimal I-reduct*, *proper reduction* etc.; in other words, we define the *normal order* of proof reduction.

In Section 4, Harrop's theorem is proved in our framework.

In Section 5, **NJ** is discussed.

In Sections 6 and 7, some further topics are discussed very roughly (informally). In Section 6, the optimality of proof reduction is pointed out. In Section 7, we relate the proper reduction with the modified realization.

In Sections 8 and 9, we present an experimental proof-checker-reducer system based on the principles of Sections 1-7. In Section 8, our *proof description language* is presented, and in Section 9, the reducer of the system is explained.

In this paper, we assume that the readers have the fundamental knowledge of the classical lambda calculus (Curry and Feys [5], Barendregt [1]) and Prawitz' natural deduction systems.

§ 2. Minimal Systems

Here we reformulate the minimal logic **NM** of Prawitz, first for the pure predicate calculus and then for the first-order intuitionistic arithmetic i.e. Heyting's arithmetic **HA**.

Our formulation is similar to that of typed p-terms of Goad, but ours is more faithful to the original natural deduction (or is the natural deduction itself) and is also carefully designed so that the following arguments will work

smoothly.

2.1. Minimal Predicate Calculus MPC

We assume that we are given the following sets of symbols:

- the countably infinite set of (individual) variables,
- the set of n-ary function symbols, for each $n \geqq 0$,
- the set of n-ary predicate symbols, for each $n \geqq 0$.

Terms and formulas are constructed from these symbols with logical connectives in the usual manner. Terms are defined as follows:

(T1) If x is a variable, then x is a term.

(T2) If f is an n-ary function symbol and $t_1, \cdots, t_n$ are terms, then $ft_1 \cdots t_n$ is a term. Informally we write $f(t_1, \cdots, t_n)$ when $n \geqq 1$.

The informal notation $f(t_1, \cdots, t_n)$ in (T2) is considered to be replaced by its formal counterpart $ft_1 \cdots t_n$ in formal metamathematical arguments. In this paper, we often introduce such informal notations to help the understanding of the reader. In some cases, the most popular notation will be regarded as an informal notation. Now the definition of formulas is:

(F1) If P is an n-ary predicate symbol and $t_1, \cdots, t_n$ are terms, then $Pt_1 \cdots t_n$ or informally $P(t_1, \cdots, t_n)$ for $n \geqq 1$ is a formula, especially called an atomic formula.

(F2) If A, B are formulas, then $\wedge AB$, $\vee AB$, $\supset AB$ are formulas. Informally, but always, we write $A \wedge B$, $A \vee B$, $A \supset B$ respectively.

(F3) If x is a variable and A is a formula, then $\forall xA$, $\exists xA$ are formulas.

We use the usual round parentheses (,) for grouping terms or formulas, but remember they are another kind of informal notation. The order of precedence among the logical connectives is $\forall$, $\exists$, $\wedge$, $\vee$, $\supset$ with $\forall$ having the highest precedence and $\supset$ the lowest. As binary operators, $\wedge$, $\vee$ and $\supset$ associate to the right.

Let us summarize the syntactic variables:

- x, y, z for variables,
- f, g, h for function symbols,
- P, Q, R for predicate symbols,
- r, s, t for terms,
- A, B, C for formulas.

They may be indexed or have primes.

Free and bound occurrences of variables in a formula are defined as usual. Those formulas which only differ in their naming of bound variables are considered to be identical.

The notations for substitution:

(S1) $s_x[t]$ denotes the result of substituting t for all the occurrences of x in s.

(S2) $A_x[t]$ denotes the result of substituting t for all the free occurrences of x in A, possibly with some changes of bound variables of A to avoid the variables of t being bound in A.

In order to define proofs of **MPC**, we assume that we have

- a countably infinite set of symbols for each formula A, whose element is called an assumption of A,

and for each A, B, C, x, t the following symbols:

$$\wedge_I{}^{A\wedge B},\ \wedge_{EL}{}^{A},\ \wedge_{ER}{}^{B},\ \vee_{IL}{}^{A\vee B},\ \vee_{IR}{}^{A\vee B},\ \vee_E{}^{C},$$
$$\supset_I{}^{A\supset B},\ \supset_E{}^{B},\ \forall_I{}^{\forall xA},\ \forall_E{}^{A_x[t]},\ \exists_I{}^{\exists xA},\ \exists_E{}^{C}$$

α^A, β^A, γ^A, α, β, γ, possibly indexed or with primes, are used to denote an assumption of A. The superfixed formulas of the symbols $\wedge_I{}^{A}$, $\cdots$ may also be omitted informally. $\wedge_I$, $\vee_{IL}$, $\vee_{IR}$, $\supset_I$, $\forall_I$, $\exists_I$ are called introduction symbols; $\wedge_{EL}$, $\wedge_{ER}$, $\vee_E$, $\supset_E$, $\forall_E$, $\exists_E$ are called elimination symbols. Now the definition of proofs goes as follows; Π, Σ etc. denote proofs:

(P1) α^A is a proof of A. Informally we write

$$[A]_\alpha,\quad [A],\quad A$$

etc. for α^A.

(P2) If Π is a proof of A and Σ is a proof of B, then $\wedge_I{}^{A\wedge B}\Pi\Sigma$ is a proof of $A\wedge B$. Informally we write

$$\frac{\Pi\ \Sigma}{A\wedge B}\wedge_I,\qquad \frac{\Pi\ \Sigma}{A\wedge B}$$

etc. instead of $\wedge_I\Pi\Sigma$. When Π is a proof of A, we also write

$$\begin{matrix}\Pi\\ A\end{matrix}$$

for Π to show A explicitly. Using this, $\wedge_I\Pi\Sigma$ may be written as

$$\frac{\begin{matrix}\Pi & \Sigma\\ A & B\end{matrix}}{A\wedge B}\wedge_I$$

etc.

(P3) If Π is a proof of $A\wedge B$, then $\wedge_{EL}{}^{A}\Pi$ or informally

$$\frac{\begin{matrix}\Pi\\ A\wedge B\end{matrix}}{A}\wedge_{EL}$$

etc. is a proof of A.

(P4) If Π is a proof of $A\wedge B$, then $\wedge_{ER}{}^{B}\Pi$ or informally

$$\frac{\begin{matrix}\Pi\\ A\wedge B\end{matrix}}{B}\wedge_{ER}$$

etc. is a proof of B.

(P5) If Π is a proof of A, then $\vee_{IL}{}^{A\vee B}\Pi$ or informally

$$\frac{\begin{matrix}\Pi\\ A\end{matrix}}{A\vee B}\vee_{IL}$$

etc. is a proof of $A \vee B$.

(P6) If Π is a proof of B, then $\vee_{IR}{}^{A \vee B}\Pi$ or informally

$$\dfrac{\begin{matrix}\Pi\\ B\end{matrix}}{A \vee B} \vee_{IR}$$

etc. is a proof of $A \vee B$.

(P7) If Π is a proof of $A \vee B$ and Σ_1, Σ_2 are proofs of C, then $\vee_E{}^C \Pi \alpha^A \Sigma_1 \beta^B \Sigma_2$ or informally

$$\dfrac{\begin{matrix}\Pi & [A] & [B]\\ A \vee B & \Sigma_1 & \Sigma_2\\ & C & C\end{matrix}}{C} \vee_E$$

etc. is a proof of C.

(P8) If Π is a proof of B, then $\supset_I{}^{A \supset B}\alpha^A \Pi$ or informally

$$\dfrac{\begin{matrix}[A]\\ \Pi\\ B\end{matrix}}{A \supset B} \supset_I$$

etc. is a proof of $A \supset B$.

(P9) If Π is a proof of $A \supset B$ and Σ is a proof of A, then $\supset_E{}^B \Pi \Sigma$ or informally

$$\dfrac{\begin{matrix}\Pi & \Sigma\\ A \supset B & A\end{matrix}}{B} \supset_E$$

etc. is a proof of B.

(P10) If Π is a proof of A and x, Π satisfy the condition on variables (c.o.v. for short, which will be defined later), then $\forall_I{}^{\forall x A} x \Pi$ or informally

$$\dfrac{\begin{matrix}\Pi\\ A\end{matrix}}{\forall x A} \forall_I$$

etc. is a proof of $\forall x A$.

(P11) If Π is a proof of $\forall x A$, then $\forall_E{}^{A_x[t]} \Pi t$ or informally

$$\dfrac{\begin{matrix}\Pi\\ \forall x A\end{matrix}}{A_x[t]} \forall_E$$

etc. is a proof of $A_x[t]$.

(P12) If Π is a proof of $A_x[t]$, then $\exists_I{}^{\exists x A} t \Pi$ or informally

$$\dfrac{\begin{matrix}\Pi\\ A_x[t]\end{matrix}}{\exists x A} \exists_I$$

etc. is a proof of $\exists x A$.

(P13) If Π is a proof of $\exists x A$, Σ is a proof of C, x, α^A, Σ satisfy the c.o.v. and x does not occur free in C, then $\exists_E{}^C \Pi x \alpha^A \Sigma$ or informally

$$\dfrac{\begin{matrix}\Sigma & [A]\\ \exists x A & \Sigma\\ & C\end{matrix}}{C} \exists_E$$

etc. is a proof of C.

For improving readability, we sometimes use (,) to group proofs.

When Π is a proof of A, A is called the end formula of Π.

Free and bound (or open and closed) occurrences of assumptions in a proof are determined as follows:

(i) α is said to occur free in α itself.

(ii) In $\vee_E\Pi\alpha\Sigma_1\beta\Sigma_2$, a free occurrence of α in Σ_1 becomes bound by the occurrence of α immediately preceding Σ_1.

(iii) In $\vee_E\Pi\alpha\Sigma_1\beta\Sigma_2$, a free occurrence of β in Σ_2 becomes bound by the occurrence of β immediately preceding Σ_2.

(iv) In $\subset_I\alpha\Pi$, a free occurrence of α in Π becomes bound by the occurrence of α immediately preceding Π.

(v) In $\exists_E\Pi x\alpha\Sigma$, a free occurrence of α in Σ, becomes bound by the occurrence of α immediately preceding Σ.

(vi) In the other cases, a free (or bound) occurrence of an assumption in a sub-proof remains free (or bound) in the whole proof.

Free and bound occurrences of variables in a proof are determined as follows:

(i) Those variables which occur free (or bound) in A are said to occur free (or bound) in α^A or in a proof headed by $\wedge_I{}^A, \cdots$.

(ii) Those variables which occur in t are said to occur free in $\forall_E\Pi t$, $\exists_I t\Pi$.

(iii) In $\forall_I x\Pi$, a free occurrence of x in Π becomes bound by the occurrence of x immediately preceding Π.

(iv) In $\exists_E\Pi x\alpha\Sigma$, a free occurrence of x in α or Σ becomes bound by the occurrence of x immediately preceding α.

(v) In the other cases, a free (or bound) occurrence of a variable in a sub-proof remains free (or bound) in the whole proof.

Those proofs which only differ in their naming of bound assumptions (of the same formula) or of bound variables are considered to be identical.

x, Π are said to satisfy the condition on variables (c.o.v. for short) iff for each free α^A in Π, x does not occur in A. x, α, Π are said to satisfy the c.o.v. iff for each free β^B in Π other than α, x does not occur in B.

The notations for substitution:

(S3) Provided that Π is a proof of A, $\Sigma_{\alpha^A}[\Pi]$ denotes the result of substituting Π for all the free occurrences of α^A in Σ, possibly with some changes of bound assumptions or variables of Σ. Informally we write

$$\begin{matrix}\Pi\\ [A]\\ \Sigma\end{matrix}$$

etc. instead of $\Sigma_{\alpha^A}[\Pi]$.

(S4) $\Sigma_x[t]$ denotes the result of substituting t for all the free occurrences of x in Σ, possibly with some changes of bound variables of Σ. In substituting t for x, each assumption α^A (or α of A) of Σ is replaced by an assumption of $A_x[t]$ which does not occur in Σ and is unique to α^A (or α); *par abus de langage*, we use $\alpha^{A_x[t]}$ (or just α) to denote this new assumption. $\wedge_I{}^A, \cdots$ are replaced by $\wedge_I{}^{A_x[t]}, \cdots$.

$\Pi\vdash A$ denotes that Π is a proof of A.

2.2. Reduction of MPC

The binary relation $\rightarrow_1$ (called the one-step reducibility) between proofs is defined as follows:

(R1) $\wedge_{EL}(\wedge_I \Pi\Sigma)\rightarrow_1 \Pi$, or informally

$$\dfrac{\dfrac{\overset{\Pi}{A}\ \overset{\Sigma}{B}}{A\wedge B}}{A} \rightarrow_1 \overset{\Pi}{A}$$

(R2) $\wedge_{ER}(\wedge_I \Pi\Sigma)\rightarrow_1 \Sigma$

(R3) $\vee_E(\vee_{IL}\Pi)\alpha\Sigma_1\beta\Sigma_2\rightarrow_1 \Sigma_{1\alpha}[\Pi]$

(R4) $\vee_E(\vee_{IR}\Pi)\alpha\Sigma_1\beta\Sigma_2\rightarrow_1 \Sigma_{2\beta}[\Pi]$

(R5) $\supset_E(\supset_I \alpha\Pi)\Sigma\rightarrow_1 \Pi_\alpha[\Sigma]$

(R6) $\forall_E(\forall_I x\Pi)t\rightarrow_1 \Pi_x[t]$

(R7) $\exists_E(\exists_I t\Pi)x\alpha\Sigma\rightarrow_1 \Sigma_x[t]_\alpha[\Pi]$

(R8) If $\Pi\rightarrow_1\Pi'$ and Π is a sub-proof (occurrence) of Σ, then $\Sigma\rightarrow_1\Sigma'$, where Σ' is the result of replacing Π by Π' in Σ.

$\rightarrow$ is defined to be the reflexive, transitive closure of $\rightarrow_1$.

The words *redex, reduction, normal* etc. are defined as in $\boldsymbol{\lambda\beta}$**K**-calculus ($\boldsymbol{\lambda\beta}$**K** for short).

A redex (occurrence) of a proof is its sub-proof (occurrence) of the form that is one of the left hand sides of $\rightarrow_1$ in (R1)-(R7). Note that in **MPC**, a redex takes the form

$$EI\cdots,$$

where E is an elimination symbol and I is an introduction symbol. We use Δ etc. to denote redexes.

A one-step reduction is identified with its corresponding redex, i.e. the redex which is reduced by (R1)-(R7) in that reduction. Thus we also use Δ etc. to denote one-step reductions.

A reduction is considered to be a sequence of one-step reductions, which in turn can be regarded as a sequence of redexes. (The earlier redex is written to the left.) We use σ, τ etc. to denote reductions. $\sigma:\Pi\rightarrow\Pi'$ denotes that σ is a reduction from Π to Π'. $\Pi\rightarrow\Pi'$ means that $\sigma:\Pi\rightarrow\Pi'$ for some σ. $\sigma\tau$ is the composition (or concatenation) of σ and τ i.e. a reduction which first applies σ and then τ.

A proof is normal iff it has no redex.

The position of a redex in a proof is identified with that of the left-most symbol of the redex.

2.3. Heyting's Arithmetic HA

HA is constructed from **MPC** by the following modifications.

The function and predicate symbols represent computable (or say primitive recursive) arithmetical functions and predicates; they include

- 0 as a 0-ary function symbol, representing the natural number 0,
- S as a unary function symbol, representing the successor function,
- $=$ as a binary predicate symbol, representing the equality predicate.

We always write $s=t$ for $=st$.

For the time being, however, we restrict ourselves to the case that there are only two function symbols i.e. 0 and S.

The clauses (P1)-(P13), (R1)-(R8) are augmented by

(P14) If $\Pi\vdash A_x[0]$, $\Sigma\vdash A_x[Sx]$ and x, α^A, Σ satisfy the c.o.v., then $\omega^{A_x[t]}t\Pi x\alpha\Sigma\vdash A_x[t]$. The informal notation is

$$\frac{\overset{\Pi}{A_x[0]} \quad \overset{[A]}{\overset{\Sigma}{A_x[S_x]}}}{A_x[t]}$$

(R9) $\omega 0\Pi x\alpha\Sigma\to_1\Pi$

(R10) $\omega St\Pi x\alpha\Sigma\to_1\Sigma_x[t]_\alpha[\omega t\Pi x\alpha\Sigma]$.

The modifications concerning free and bound occurrences of variables and assumptions are left to the reader.

§ 3. Minimal ∃-Reduct and Proper Reduction

A proof is said to be existential if its end formula is an existential formula.

Let Π be existential say $\Pi\equiv\underset{\exists xA}{\Pi}$. (In this paper, $\equiv$ denotes the syntactical identity in general.) If

$$\Pi\to\Sigma\equiv\frac{\overset{\Sigma'}{A_x[t]}}{\exists xA}\equiv\exists_I t\Sigma',$$

Σ is called an ∃-reduct of Π. Π is said to be ∃-reducible if it has an ∃-reduct. A minimal ∃-reduct of Π is an ∃-reduct $\Pi_\exists$ of Π such that for any ∃-reduct Σ of Π, there exists $\sigma:\Pi_\exists\to\Sigma$ i.e. $\Pi\to\Pi_\exists\to\Sigma$.

Since the strong normalization theorem holds in **MPC** (or **HA**), $\to$ is a partial order between proofs. Reading $\to$ as $\leqq$, the minimal ∃-reduct of an existential proof is really the minimal element of all the ∃-reducts of the proof; in other words, it is the nearest ∃-reduct to the original proof. We want to prove:

Every ∃-reducible existential proof has a minimal ∃-reduct.

To expose the meaning of the theorem, we first prove its counterpart in $\lambda\beta$**K**.

3.1. Minimal λ-reduct and Proper Reduction of λβK

We use M, N etc. to denote λ-terms. We also use $\to_1$, $\to$ etc. for $\lambda\beta$**K** as for **MPC** or **HA**.

If $M\to N\equiv\lambda u.N'$, N is called a λ-reduct of M. M is said to be λ-reducible

if it has a λ-reduct. A minimal λ-reduct of M is a λ-reduct M_λ of M such that for any λ-reduct N of M, there exists $\sigma: M_\lambda \to N$ i.e. $M \to M_\lambda \to N$.

Let $M \equiv (\lambda u.M_0)M_1 \cdots M_n$ $(n \geqq 1)$. We call $(\lambda u.M_0)M_1$ a proper head redex (or just a proper redex) of M. A proper reduction is a reduction consisting of only successive proper redexes.

A proper reduction is said to be maximal if no further proper reduction is possible. A proper reduction ending in a λ-reduct is maximal. A maximal proper reduction from a λ-term is unique.

Usually a proper reduction is called *call-by-name* and it forms an initial segment of the so-called normal order or the head reduction.

Theorem 3.1. *If M is λ-reducible, then M has a minimal λ-reduct.*

Proof. Let $\sigma: M \to N \equiv \lambda u.N'$. By the standardization theorem of $\boldsymbol{\lambda\beta K}$, there exists a standard reduction $\sigma_S: M \to N$. σ_S can be split as

$$\sigma_S = \sigma_p \sigma_i ,$$

where σ_p is proper and σ_i contains no proper redex. Let $\sigma_p: M \to M_\lambda$. Since $N \equiv \lambda u.N'$, M_λ must be of the form $\lambda u.M_\lambda'$. This means that M_λ is a λ-reduct of M and $M \to M_\lambda \to N$. By the remark before the theorem, this M_λ is independent of N. Therefore M_λ is also a minimal λ-reduct of M. □

The above proof tells us that if M is λ-reducible, the maximal proper reduction starting from M will end in a minimal λ-reduct.

3.2. Proper Reduction of MPC and HA

If $\Pi \to \Sigma$ and Σ is headed by an introduction symbol, then Σ is called an I-reduct of Π. The notions I-reducible, minimal I-reduct are defined as before.

Let

$$\Pi \equiv E_1 \cdots E_n \Delta \cdots ,$$

where $E_1, \cdots, E_n$ $(n \geqq 0)$ are elimination symbols and Δ is the left-most redex of Π. Then Δ is called a proper redex of Π. A proper reduction is one consisting of only proper redexes as in $\boldsymbol{\lambda\beta K}$.

A standard reduction of **MPC** (or **HA**) is one which just proceeds from left to right.

We can prove the standardization theorem for **MPC** and **HA** by the same technique as for $\boldsymbol{\lambda\beta K}$. If $\sigma_S: \Pi \to \Sigma$ is standard, σ_S splits as

$$\sigma_S = \sigma_p \sigma_i ,$$

where σ_p is proper and σ_i contains no proper redex. If σ_i is not empty, it is composed of standard reductions of the sub-proofs of Π', where $\sigma_p: \Pi \to \Pi'$.

Now, we can prove

Theorem 3.2. *If Π is I-reducible, then Π has a minimal I-reduct.*

§ 4. Harrop's Theorem

A sub-proof occurrence Σ in Π is said to be safe, iff Π is not of the form

$$\Pi \equiv E_1 \cdots E_n \Sigma \cdots,$$

where $E_1, \cdots, E_n$ $(n \geqq 0)$ are elimination symbols. Σ is unsafe iff it is not safe.

A Harrop formula is defined as follows:

(H1) If A is atomic, then A is Harrop.

(H2) If A and B are Harrop, then $A \wedge B$ is Harrop.

(H3) If B is Harrop, then $A \supset B$ is Harrop, where A is arbitrary.

(H4) If A is Harrop, then $\forall x A$ is Harrop.

A formula is non-Harrop iff it is not Harrop.

A proof is called Harrop (or non-Harrop), if its end formula is Harrop (or non-Harrop).

Lemma 4.1. *Every Harrop sub-proof of a non-Harrop proof* Π *is safe.*

Proof. By induction on $|\Pi|$.

If Π is an assumption or the first symbol of Π is an introduction symbol, then the only unsafe sup-proof of Π is Π itself. But Π is non-Harrop, so there is nothing to prove.

If the first symbol is $\wedge_{EL}$,

$$\Pi \equiv \wedge_{EL}\Pi' \equiv \frac{\begin{matrix}\Pi' \\ A \wedge B\end{matrix}}{A}.$$

Since A is non-Harrop, so is $A \wedge B$. By induction hypothesis, every Harrop sub-proof of Π' is safe. Since a Harrop sub-proof of Π should appear in Π', every Harrop sub-proof of Π is safe.

The other cases may be treated similarly. □

Here we want to mention the strong normalization theorem for **MPC** and **HA** explicitly:

Every reduction is finite.

See Prawitz [22] [23], Jervell [13] or Troelstra [25].

Theorem 4.2 (**Harrop**). *In* **MPC**, *if* Π *is non-Harrop and all the free assumptions of* Π *are Harrop, then the proper reduction starting from* Π *terminates in the minimal I-reduct of* Π.

Proof. By Lemma 4.1, all the assumptions which occur free in the course of the proper reduction are safe. If the maximal reduction from Π terminates, it should end in one of the following forms:

$$E_1 \cdots E_n \alpha \cdots \quad \text{or} \quad I \cdots.$$

($E_1, \cdots, E_n$ are elimination symbols, and I is an introduction symbol.) But the

first case is impossible by the safety of the assumptions. Since the reduction terminates, it must eventually end in the minimal I-reduct of Π. □

Theorem 4.3. *In* **HA**, *if* Π *is non-Harrop, all the free assumptions of* Π *are Harrop and no variables occur free in* Π, *then the proper reduction starting from* Π *terminates in the minimal I-reduct of* Π.

Proof. By Lemma 4.1, it is enough to prove that whenever a sub-proof of the form $\omega t \cdots$ occurs unsafe, the term t is closed. This can be easily verified by induction on $|\Pi|$. □

Usually these theorems are proved by observation of the normal form of Π: to assume that the normal form is not an I-reduct contradicts the normality. But taking the reduction into account, the proof becomes much clearer as above.

§ 5. Intuitionistic Systems

5.1. Intuitionistic Predicate Calculus IPC

IPC is given by adding to **MPC** the following symbols:

$$\bot,\quad \bot_I{}^A \quad \text{(for each } A\text{)}$$

and the clauses:

(F4) $\bot$ is a formula.

(P14) If Π is a proof of $\bot$, then $\bot_I{}^A\Pi$ or informally

$$\frac{\Pi}{\dfrac{\bot}{A}\bot_I}$$

etc. is a proof of A.

We write $\sim A$ for $A \supset \bot$.

Below, we will use the consistency of **IPC**.

Theorem 5.1. *If no assumptions occur free in* Π, *the proper reduction starting from* Π *terminates in the minimal I-reduct of* Π.

Proof. We should only prove that a sub-proof of the form $\bot_I\Sigma$ does not occur unsafe in the course of the reduction. If $\bot_I\Sigma$ occurs unsafe, no free assumptions occur in Σ. But this would contradict the consistency of **IPC**, since $\Sigma \vdash \bot$. □

5.2. HA/Π

Here we version up **HA**. First we add the symbols $\bot$, $\bot_I{}^A$ and their related clauses to **HA**, where $\bot$ represents absurdity. Next we fix a set of assumptions, whose element is called an axiom. Here, we require that an axiom be an assumption of a true closed Harrop formula. As is well-known, **HA** is consistent relative to the set of axioms.

We write $\Pi\vdash_0 A$, if $\Pi\vdash A$, no variables occur free in Π and no assumptions other than axioms occur free in Π. When $\Pi\vdash_0 A$, A is closed.

Theorem 5.2. *If Π is non-Harrop and $\Pi\vdash_0 A$, then the proper reduction starting from Π terminates in the minimal I-reduct of Π.*

Proof. Theorem 4.3 plus Theorem 5.1. □

§ 6. Optimal *I*-Reduct

Although the proper reduction gives the minimal I-reduct with respect to $\rightarrow$, it is not optimal in the sense of reduction cost. Berry and Lévy [2] and Lévy [18] discuss the optimality of reduction for recursive schemata and for λ-terms respectively. Their arguments also apply to proof reduction.

Let

$$|\Pi\rightarrow\Sigma|=\min\{|\sigma|\,;\ \sigma:\Pi\rightarrow\Sigma\}.$$

The optimal I-reduct Π_O of Π is defined as an I-reduct of Π s.t. $|\Pi\rightarrow\Sigma|\geqq|\Pi\rightarrow\Pi_O|$ for any I-reduct Σ of Π. We can prove:

Π_O is given by reducing Π by a complete reduction of σ_p, where $\sigma_p:\Pi\rightarrow\Pi_I$ is the proper reduction terminating in the minimal I-reduct Π_I of Π.

Roughly speaking, σ_p with the sharing mechanism, which is what Lévy calls a complete reduction, gives Π_O with the minimal cost. In general, Π_O does not coincide with Π_I.

Usually a shared proper reduction is called *call-by-need.*

§ 7. Modified Realization and Permutative Reduction

In this section, we fix the formal system to **HA** (of 2.3), except that we do not restrict the form of axioms.

The aim of this section is to observe what kind of reduction of realization corresponds to the proper reduction of a proof.

7.1. Modified Realization

We take the system of functionals of finite type with direct product, which we temporarily name **FT**, for the modified realization. The type structure of **FT** is defined as:

(τ1) 0 is a type.

(τ2) If σ, τ are types, so are $\sigma\rightarrow\tau$ and $\sigma\times\tau$.

We use σ, τ etc. to denote types. We assume that $\rightarrow$ and $\times$ are right-associative and $\times$ is stronger than $\rightarrow$.

The terms of each type are constructed by the application and the lambda abstraction by the usual manner. We list the set of constants below; their type

is indicated to the right of them. We assume that there is a distinct constant for each appropriate type, if the indicated type is a schema of types.

- 0: 0
- S: $0\to 0$
- π: $\sigma\to\tau\to\sigma\times\tau$
- π_1: $\sigma\times\tau\to\sigma$
- π_2: $\sigma\times\tau\to\tau$
- R: $0\to\tau\to(0\to\tau\to\tau)\to\tau$
- cond: $0\to\tau\to\tau\to\tau$
- dummy: τ

We write $\langle s, t\rangle$ for πst and

$$\textbf{if } s \textbf{ then } t_1 \textbf{ else } t_2$$

for cond st_1t_2.

$\to_1$ is defined as follows:

(F1) $(\lambda u.\, t)s\to_1 t_u[s]$

(F2) $\pi_1\langle s, t\rangle\to_1 s$

(F3) $\pi_2\langle s, t\rangle\to_1 t$

(F4) $R0t_0t_1\to_1 t_0$

(F5) $RStt_0t_1\to_1 t_1t(Rtt_0t_1)$

(F6) **if** 0 **then** t_1 **else** $t_2\to_1 t_1$

(F7) **if** St **then** t_1 **else** $t_2\to_1 t_2$

For each formula A, type(A) is defined as follows:

(t1) If A is atomic, then type $(A)=0\times 0$.

(t2) type$(A\wedge B)=$type$(A)\times$type(B)

(t3) type$(A\vee B)=0\times$type$(A)\times$type(B)

(t4) type$(A\supset B)=$type$(A)\to$type(B)

(t5) type$(\forall x A)=0\to$type(A)

(t6) type$(\exists x A)=0\times$type(A)

A term of **HA** is considered to be a term of **FT** of type 0.

Now, the modified realization of Π $(\vdash A)$, denoted by $\Pi^\dagger$, whose type is type (A), is defined as follows:

(M1) $\alpha^\dagger\equiv\alpha$, where α in the right-hand side is considered to be a variable of **FT** of type (A).

(M2) $(\wedge_I\Pi\Sigma)^\dagger\equiv\langle\Pi^\dagger, \Sigma^\dagger\rangle$

(M3) $(\wedge_{EL}\Pi)^\dagger\equiv\pi_1\Pi^\dagger$

(M4) $(\wedge_{ER}\Pi)^\dagger\equiv\pi_2\Pi^\dagger$

(M5) $(\vee_{IL}\Pi)^\dagger\equiv\langle 0, \langle\Pi^\dagger, \text{dummy}\rangle\rangle$

(M6) $(\vee_{IR}\Pi)^\dagger\equiv\langle S0, \langle\text{dummy}, \Pi^\dagger\rangle\rangle$

(M7) $(\vee_E\Pi\alpha\Sigma_1\beta\Sigma_2)^\dagger\equiv(\lambda u.$ **if** $\pi_1 u$ **then** $(\lambda\alpha.\, \Sigma_1^\dagger)(\pi_1\pi_2 u)$ **else** $(\lambda\beta.\, \Sigma_2^\dagger)(\pi_2\pi_2 u))\Pi^\dagger$

(M8) $(\supset_I \alpha\Pi)^\dagger \equiv \lambda\alpha.\,\Pi^\dagger$
(M9) $(\supset_E \Pi\Sigma)^\dagger \equiv \Pi^\dagger\Sigma^\dagger$
(M10) $(\forall_I x\Pi)^\dagger \equiv \lambda x.\,\Pi^\dagger$
(M11) $(\forall_E \Pi t)^\dagger \equiv \Pi^\dagger t$
(M12) $(\exists_I t\Pi)^\dagger \equiv \langle t,\ \Pi^\dagger\rangle$
(M13) $(\exists_E x\alpha\Sigma)^\dagger \equiv (\lambda u.\,(\lambda x\alpha.\,\Sigma^\dagger)(\pi_1 u)(\pi_2 u))\Pi^\dagger$
(M14) $(\omega t\Pi x\alpha\Sigma)^\dagger \equiv Rt\Pi^\dagger(\lambda x\alpha.\,\Sigma^\dagger)$

We assume that proofs to be realized are all closed in the following arguments.

Because of (M7) and (M13), call-by-name (or proper) reductions of Π do not correspond to call-by-name reductions of $\Pi^\dagger$. But shared reductions (i.e. reductions with structure sharing) of a proof are naturally mapped to shared reductions of its modified realization. It is expected that the call-by-need proof reduction is mapped to the usual call-by-need reduction of **FT**, but it is false again for (M13). Observing the image of the call-by-need proof reduction under †, the following facts are found:

(i) If s is of type 0, then the redex $(\lambda u.\,t)s$ is reduced only if s is a numeral (i.e. $s \equiv S\cdots S0$). When s is not a numeral, s is reduced before $(\lambda u.\,t)s$ is reduced to $t_u[s]$.

(ii) When **if** s **then** t_1 **else** t_2 is reduced, s is a numeral.

(iii) When Rst_0t_1 is reduced, s is a numeral.

(iv) When a term of type $0\times\sigma$ is reduced, the result is of the form $\langle s, t\rangle$, where s is a numeral.

We may say in a word that the reduction is call-by-value for type 0 and call-by-need for other types. ((t1) was for this.)

Restricting (F1) to (i) above, we may prove the following theorem:

Π is $\exists$-reducible iff $\pi_1\Pi^\dagger$ can be reduced to a numeral.

7.2. Permutative Reduction

Without any restriction on (F1), **FT** is stronger than **HA** in the following sense: there exists $\Pi \vdash \exists x A$ s.t. Π is not $\exists$-reducible, while $\pi_1\Pi^\dagger$ reduces to a numeral.

It is for this fact that we need the permutative reduction and the immediate simplification of proofs. We only have to consider the permutative reduction of $\exists_E$ and the immediate simplification of $\exists_E$ for this purpose. Originally, they take the forms

$$E(\exists_E \Pi x\alpha\Sigma)\cdots \to_1 \exists_E \Pi x\alpha(E\Sigma\cdots),$$

where E is an elimination symbol, and

$$\exists_E \Pi x\alpha\Sigma \to_1 \Sigma,$$

where α does not occur free in Σ. But we interpret them more computationally as follows.

What is reduced is not just a proof, but a pair of a proof and an environment; an environment is a list of strings of the form $\exists_E \Pi x \alpha$. Now the (proper) reduction proceeds as before until we encounter $\exists_E$. Then the proof must be of the form

$$E_1 \cdots E_n(\exists_E \Pi x \alpha \Sigma) \cdots,$$

where $E_1, \cdots, E_n$ $(n \geqq 0)$ are elimination symbols. We replace the whole proof by

$$E_1 \cdots E_n \Sigma \cdots,$$

and push $\exists_E \Pi x \alpha$ at the top of the environment. Then the reduction proceeds for the above proof. When the proof becomes of the form

$$E_1 \cdots E_n \alpha \cdots \quad \text{or} \quad E_1 \cdots E_n \omega x \cdots,$$

we search the environment for α or x. If $\exists_E \Pi x \alpha$ is found in the environment, we reduce Π. When we get the minimal I-reduct $\exists_I t \Pi'$ of Π, we substitute t and Π' for x and α in the original proof and the environment, delete $\exists_E x \alpha \Pi$ from the environment, and reduce

$$E_1 \cdots E_n \Pi' \cdots \quad \text{or} \quad E_1 \cdots E_n \omega t \cdots.$$

To avoid the variable (or assumption) conflict, we may need to rename x or α when we push $\exists_E \Pi x \alpha$ in the environment. Whenever the proof contains no free occurrence of α, we delete $\exists_E x \alpha \Pi$ from the environment.

The above reduction is equivalent to (or in other words, simulates) the reduction of the modified realization. And it can be translated to a combination of the permutative reductions, the immediate simplifications and the other reductions in the following sense: if Π reduces to Π' with environment $\mathbf{E}$, then $\Pi \to \mathbf{E}\Pi'$ with permutative reductions and immediate simplifications.

§ 8. Proof Description Language

In this and next sections, we present our experimental proof-checker-reducer system based on the principles developed so far. The system is implemented by the author on VAX11 under VM/Unix (Kernighan and McIlroy [14]) at Computer Centre of University of Tokyo.

In this section, we explain the proof checker of the system. In designing the checker, we paid our attention on the readability of written proofs, since proofs constructed by the so-called interactive proof checkers such as FOL (Weyhrauch [27] [28]) tend to be unstructured sequences of commands like Assembler programs. This leads to the idea of *proof description languages*, which correspond to higher level programming languages like Algol, PL/I etc. The typical one is PL/CV (Constableand O'Donnel [4]).

At the same time, we wanted to base the language on our formulation of natural deduction (§ 2 and § 5). Since symbols like $\supset_I$, $\vee_E$, $\forall_I$ etc. have *binding* occurrences of assumptions or variables, it is natural to regard them as state-

ment symbols, while other symbols may be operators for constructing expressions or *proof-expressions*. They have turned very natural and easy to write when we get used to them.

In the following explanations,

$$[\cdots] \equiv ——$$

means that the construct ··· of the language corresponds to the construct —— of Section 2 or Section 5.

We will not give precise syntax of the language using BNF etc.

8.1. Primitive Symbols

We divide the ascii characters as follows:

alphanumeric characters
    abcdefghijklmnopqrstuvwxyz
    ABCDEFGHIJKLMNOPQRSTUVWXYZ
    0123456789
delimiters
    ( ) [ ] { } , ; : .
other characters
    other graphic characters
white characters
    space tab newline

A name is a list of alphanumeric strings, which are separated by one or more white characters to one another. E.g.

Theorem 11
x
this is a name.

Any string in a name should not coincide with a keyword. A name should not coincide with a number.

A number is a nonnegative integer expressed in the usual decimal notation.

Names are used as variables, function symbols, predicate symbols, assumptions or labels (which will be introduced later).

A keyword is one of the following strings:

IF ANY CASE LEFT RIGHT SINCE LET
CONTRADICTION IND BASE STEP
A E FALSE ID SYM TR EQL EQ
PREDICATE FUNCTION PREFIX POSTFIX INFIX
REDUCE EVAL ATTACH

An operator is a string of *other* characters. There are special operators and declared operators. The special operators are

& | − > ~ ^

Declared operators are used as infix predicate symbols, postfix function symbols etc. etc.

8.2. Term and Formula

Terms are constructed from variables, function symbols and ( ) and, in the usual manner. (Similarly for atomic formulas.) We use ' as a postfix function symbol representing the successor function; i.e.

$$\lceil t' \rceil \equiv St.$$

(We must write $\lceil t' \rceil \equiv S\lceil t \rceil$, but we won't be so rigorous.) Numbers are regarded as numerals; e.g.

$$\lceil 5 \rceil \equiv SSSSS0.$$

Formulas are written as follows:

$\lceil A \ \& \ B \rceil \equiv A \wedge B$
$\lceil A \mid B \rceil \equiv A \vee B$
$\lceil A -> B \rceil \equiv A \supset B$
$\lceil \mathrm{A}(x)A \rceil \equiv \forall x A$
$\lceil \mathrm{E}(x)A \rceil \equiv \exists x A$
$\lceil \mathrm{FALSE} \rceil \equiv \bot$
$\lceil \tilde{} A \rceil \equiv \sim A$

E.g.

A(x) [human(x)−>mortal(x)]

is a formula. Formulas are grouped by [ ], not by ( ).

8.3. Proof-Expression

In our language, proofs headed by $\wedge_I$, $\wedge_{EL}$ etc. are written as proof-expressions. We use π, σ etc. to denote proof-expressions.

$\lceil \pi \ \& \ \sigma \rceil \equiv \wedge_I \pi\sigma$
$\lceil \& \ \mathrm{LEFT} \ \pi \rceil \equiv \wedge_{EL} \pi$
$\lceil \& \ \mathrm{RIGHT} \ \pi \rceil \equiv \wedge_{ER} \pi$
$\lceil \mid \mathrm{LEFT} \ \pi \rceil \equiv \vee_{IL} \pi$
$\lceil \mid \mathrm{RIGHT} \ \pi \rceil \equiv \vee_{IR} \pi$
$\lceil \pi \hat{} \sigma \rceil \equiv \supset_E \pi\sigma$
$\lceil \pi(t) \rceil \equiv \forall_E \pi t$
$\lceil \mathrm{E}(t)\pi \rceil \equiv \exists_I t\pi$

The difference from Section 2 or Section 5 is that the end formula of a proof-expression is not written explicitly as a superfix of a symbol, but is determined from the context. See 8.4. Proof-expressions are grouped by ( ).

8.4. Proof-Statement

Proofs headed by $\supset_I$, $\vee_E$, $\forall_I$ etc. become proof-statements. (We use Π, Σ for proof-statements.) E. g.

IF (: α) Π

is a proof-statement and corresponds to $\supset_I \alpha \Pi$. A formula may precede a proof-statement; e. g.

$A \to B$ IF (: α) Π,

which asserts that the end formula of IF(: α)Π is $A \to B$. From this it is determined that α is an assumption of A and Π is a proof of B. We may also put a formula before: α to explicitly state the end formula of α. We can write

IF (A: α) $B\Pi$

The end formula of this proof-statement is $A \to B$. (A: α) or (: α) are called assumption-declarations and used as binding occurrences of assumptions.

The checker of the system, using the arguments as above, checks whether the end formula of each part of the given proof is determined uniquely.

If π is a proof-expression,

: π ;

is a proof-statement. (Obviously, ⟦: π ;⟧≡⟦π⟧.) The other proof-statements are listed below:

⟦IF(: α)Π⟧≡$\supset_I \alpha \Pi$
⟦ANY(x)Π⟧≡$\forall_I x \Pi$
⟦CASE Π LEFT (: α) Σ_1 RIGHT (: β) Σ_2⟧≡$\vee_E \Pi \alpha \Sigma_1 \beta \Sigma_2$
⟦SINCE Π LEFT (x)(: α)Σ⟧≡$\exists_E \Pi x \alpha \Sigma$
⟦CONTRADICTION Π⟧≡$\perp_I \Pi$
⟦IND(x) BASE Π STEP (: α) Σ⟧≡$\forall_I x \omega x \Pi x \alpha \Sigma$

Proof-statements may be grouped to form a proof-clause. It is of the form

{
Π_1
Π_2
⋮
Π_n
}

The end formula of a proof-clause is considered to be the end formula of its last proof-statement. Proof-statements in a proof-clause (or in the top-level) may have a label, by which they are referred to in the rest of the proof-clause (or in the succeeding inputs). A label is a name and is placed before a proof-statement separated by a period.

We now give an example. We prove that Socrates is mortal from the axioms that every man is mortal and that Socrates is a man.

```
PREDICATE human (1);
PREDICATE mortal (1);
FUNCTION Socrates (0);

Axiom 1.
    A(x) [human (x)->mortal (x)];

Axiom 2.
    human (Socrates);

Theorem.
    mortal (Socrates)
    : Axiom 1 (Socrates) ^ Axiom 2;
```

The first 3 lines declare symbols human, mortal and Socrates. The numbers in parentheses are the arities of the symbols. The proofs labeled Axiom 1 and Axiom 2 are empty (or null), so they are assumed as axioms.

8.5. Equality

$=$ is an infix predicate symbol. Rules concerning $=$ are not discussed in Section 2 or Section 5, but included here, because they are indispensable for actually writing proofs. They are summarized as follows:

ID$\vdash t=t$
$\pi\vdash t=s\Rightarrow$SYM $\pi\vdash s=t$
$\pi\vdash t_1=t_2$, $\sigma\vdash t_2=t_3\Rightarrow\pi$ TR $\sigma\vdash t_1=t_3$
$\pi\vdash t_1=t_2\Rightarrow$EQL $\pi\vdash s_x[t_1]=s_x[t_2]$
$\pi\vdash A_x[s]$, $\sigma\vdash t=s\Rightarrow\pi$ EQ $\sigma\vdash A_x[t]$.

8.6. Examples

Now, we begin arithmetic.

```
Successor is nontrivial.
    A(x)~0=x';

Theorem 1.
    A(x) [x=0|E(y)x=y']
    IND(x)
    BASE
        0=0|E(y) 0=y' {
        L1.
            0=0: ID;
        L2.
            0=0|E(y) 0=y': |LEFT L1;
        }
```

```
    STEP (: IH)
        x'=0|E(y) x'=y' {
        L1.
            x'=x': ID;
        L2.
            E(y) x'=y': E(x) L1;
        L3.
            : |RIGHT L2;
        }

Theorem 2.
    A(x) [E(y) x=y'->~x=0]
    ANY(x)
        IF(E(y) x=y': A0)
            SINCE
                E(y) x=y': A0;
            LET(y) (x=y': A1)
                IF(x=0: A2) {
                L1.
                    0=y': SYM A2 TR A1;
                L2.
                    ~0=y'
                    : Successor is nontrivial (y);
                L3.
                    FALSE: L2 ^ L1;
                }

Theorem 3.
    A(x) [x=0|~x=0]
    ANY(x)
        CASE
            x=0|E(y) x=y'
            : Theorem 1(x);
        LEFT (x=0: L)
            : |LEFT L;
        RIGHT (E(y) x=y': R)
            : |RIGHT (Theorem 2(x) ^ R);

Theorem 4.
    A(x) E(y) [x=0 & y=0|~x=0 & x=y']
    ANY(x)
        CASE
            x=0|E(y) x=y': Theorem 1(x);
```

```
LEFT (x=0: L) {
    L1.
        x=0: L;
    L2.
        0=0: ID;
    L3.
        x=0 & 0=0: L1 & L2;
    L4.
        : E(0) |LEFT L3;
    }
    RIGHT (E(y) x=y': R)
    SINCE
        : R;
    LET(y) (x=y': A1) {
    L1.
        ~x=0 & x=y'
        : (Theorem 2(x) ^ R) & A1;
    L2.
        : E(y)|RIGHT L1;
    }
```

§ 9. Reducer

9.1. REDUCE

The reducer of the system is invoked by the following command:

```
REDUCE;
```

Let us give an example. Assume that we have already input the examples in 8.6. The following session may follow:

```
: Theorem 3(3);
3=0|~3=0
REDUCE;
~3=0
```

The outputs from the system are written in italics. The first line is a proof (or proof-statement) of 3=0|~3=0. The system replies by typing the formula proved. The third line invokes the reducer. It reduces the last proof and prints the result. In this case, its I-reduct is of the form |RIGHT π i.e. $\vee_{IR}\pi$, so the system printed the end formula of π, ~3=0.

The proof to be reduced should be of a disjunctive formula or of an existential formula.

9.2. ATTACH

HA in Section 2 or Section 5 has only two function symbols, but in the actual system, we may have arbitrary function symbols and moreover we may attach a proof to them.

Theorem 4 in 8.6 is of the form

```
Theorem 4.
    A(x) E(y) [x=0 & y=0|~x=0 & x=y']
        ...
        proof
        ...
```

Then

```
FUNCTION P(1);
Predecessor definition.
    A(x) [x=0 & P(x)=0|~x=0 & x=P(x)'];
ATTACH P: Theorem 4, Predecessor definition;
```

attach the above proof (i.e. Theorem 4) to the function symbol P. The axiom labeled Predecessor definition above is the defining axiom of P. How the attachment works will be explained in 9.4.

The syntax is

```
ATTACH f: l1, l2;
```

where f is a function symbol and l_1 and l_2 are labels. When f is n-ary, l_1 should be a label of a proof of

$$A(x_1) \cdots A(x_n)\ E(y)\ [\cdots y \cdots]$$

and l_2 should be of

$$A(x_1) \cdots A(x_n)\ [\cdots f(x_1, \cdots, x_n) \cdots].$$

To keep the system consistent, we require that f be unattached and that the proof attached to f be f-free.

Π is said to be f-free, if f does not occur in t in a sub-proof of Π of the form $\forall_E \Pi' t$, $\exists_I t \Pi'$ or $\omega t \Pi' x \alpha \Pi''$.

9.3. EVAL

```
EVAL t;
```

It just reduces (or evaluates) the term t by the evaluator which is a part of the reducer, and prints the value. E.g.

```
EVAL P(2);
1
```

9.4. Reduction Procedure

The difference from Section 2 or Section 5 is that in a proof may occur function symbols, to which a proof may have been attached, and the additional symbols like ID, TR etc.

The principle of reduction is that terms in a proof are reduced (or evaluated) by call-by-value and other parts of a proof are reduced by call-by-need. More precisely, when

$$\forall_I \Pi t$$
$$\exists_I t \Pi$$
$$\omega t \Pi x \alpha \Sigma$$

occur unsafe, the term t is reduced to a numeral at once. See 7.1.

The reduction of a term proceeds as follows. If t is a numeral, then it is returned. If t is a variable, then the reduction fails. If $t \equiv f(t_1, \cdots, t_n)$ and Π_f is attached to f, first $t_1, \cdots, t_n$ are reduced to numerals (say $k_1, \cdots, k_n$) and then $\forall_E \cdots \forall_E \Pi k_1 \cdots k_n$ is reduced to its I-reduct $\exists_I n \Pi$. This n is returned for the reduction of t. If f is unattached, the reduction fails.

When ID, TR, etc. occur unsafe, the reduction fails.

To implement a complete structure sharing is very difficult and inefficient. In this system, call-by-need is implemented by the usual environment technique, augmented by the technique of lazy cons (Friedman and Wise [7]). This means that if $\wedge_I \Pi \Sigma$ is shared, Π and Σ are also shared, i.e. they are reduced only once if ever reduced. Note that $\wedge_I$ corresponds to CONS of LISP.

Conclusion

Many of the concepts and the techniques of the (classical) lambda calculus can be applied directly to the natural deduction. This paper is one of such works. Further applications are expected to bear further results; e.g. What is a λI-proof? What is a fixed point proof? etc. etc.

The reducer is a small part of our system. The implementation of the reducer was very easy, once the theory had been established. We think that it is not a very difficult task to execute a constructive proof, or to extract a program from its existence proof.

The most difficult is to actually write proofs and input them to a machine. Theorem provers *will* help us, but they are not almighty; their computational limitation is obvious. After all, we people have to write most part of proofs. Thus what is needed is to design a neat language, which is easily readable as well as writable, and to use it and gain experiences through it. The knowledge obtained in the study of programming languages will be of much use.

Our system is very small, but is a model of larger systems, in which we can write long (and practical) proofs and execute them, if they are constructive.

References

[1] Barendregt, H.P., *The lambda calculus: its syntax and semantics*, North-Holland, Amsterdam, 1981.

[2] Berry, G. and Lévy, J.-J., Minimal and optimal computations of recursive programs, *J. ACM*, **26** (1979), 148-175.

[3] de Brujin, N.G., The mathematical language AUTOMATH, its usage, and some of its extensions *Lecture Notes in Math.* 125, Springer-Verlag, 1970, 29-61.

[4] Constable, R.L. and O'Donnel, M.J., *A Programming Logic*, Winthrop, 1978.

[5] Curry, H.B. and Feys, R., *Combinatory logic* **1**, North-Holland, Amsterdam, 1968.

[6] Diller, J., Modified realization and the formulae-as-types notion, *Festschrift on the occasion of H.B. Curry's 80th birthday*, Academic Press, New York, San Francisco, London, 1980, 491-502.

[7] Friedman, D.P. and Wise, D.S., CONS should not evaluate its arguments, *Automata Languages and Programming*, Edinburgh Press, 1976, 289-296.

[8] Gentzen, G., *The collected papers of Genhard Gentzen* (*Szabo M.E. ed.*), North-Holland, Amsterdam, 1969.

[9] Goad, C.A., *Computational uses of the manipulation of formal proofs*, Ph.D. Thesis, Stanford Univ., 1980.

[10] ———, Proofs as descriptions of computation, *5th Conference on Automated Deduction Les Arcs, France, 1980, Lecture Notes in Computer Science*, **87**, Springer-Verlag, 1980, 39-52.

[11] Gödel, K., Uber eine bisher noch nicht benutzte Erweiterung des finiten Standpunktes, *Dialectica*, **12** (1958), 280-287.

[12] Howard, W.A., The formulae-as-types notion of construction, *Festschrift on the occasion of H.B. Curry's 80th birthday*, Academic Press, New York, San Francisco, London, 1980, 479-490.

[13] Jervell, H., A normalform in first order arithmetic, *Proc. Second Scandinavian Logic Symposium*, North-Holland, Amsterdam, 1970, 93-108.

[14] Kernighan, B.W. and McIlroy, M.D., *Unix Programmer's Manual, Seventh Edition, Virtual VAX-11 Version*, Bell Laboratories, 1979.

[15] Kleene, S.C., On the interpretation of intuitionistic number theory, *J. Symbolic Logic*, **10** (1945), 109-124.

[16] ———, *Introduction to metamathematics*, North-Holland, Amsterdam, 1952.

[17] Kreisel, G., Interpretation of analysis by means of constructive functionals of finite type, *Constructivity in Mathematics*, North-Holland, Amsterdam, 1959, 101-128.

[18] Lévy, J.-J., Optimal reductions in the lambda-calculus, *Festschrift on the occasion of H.B. Curry's 80th birthday*, Academic Press, New York, San Francisco, London, 1980, 159-192.

[19] Martin-Löf, P., An intuitionistic theory of types, predicative part, *Logic colloquium '73*, North-Holland, Amsterdam, 1975, 73-118.

[20] Mints, G., On E-theorems (Russian), *Investigation in constructive mathematics* VI., *Zapisky Nauk. Sem., Lenigrad, Steklov Inst.*, **40** (1974), 110-118.

[21] Nelson, D., Recursive functions and intuitionistic number theory, *Trans. Amer. Math. Soc.*, **61** (1947), 307-368.

[22] Prawitz, D., *Natural deduction*, Almquist and Wiksell, Stockholm, 1965.

[23] ———, Ideas and results in proof theory, *Proc. Second Scandinavian Logic Symposium*, North-Holland, Amsterdam, 1970,235-308.

[24] Scott, D., Constructive validity, *Lecture Notes in Math.*, **125**, Springer-Verlag, 1970, 237-275.

[25] Troelstra, A.S., *Metamathematical investigations of intuitionistic arithmetic and analysis, Lecture Hotes in Math.*, **344**, Springer-Verlag, 1973.

[26] de Vrijer, R., Big trees in λ-calculus with λ-expressions as types, *Lecture Notes in Computer Science*, **37**, Springer-Verlag, 1975, 252-271.

[27] Weyhrauch, R.W., *A Users Manual for FOL, Stanford Artificial Intelligence Laboratory Memo AIM-235.1*, Stanford University, Stanford, 1977.

[28] ———, Prolegomena to a theory of mechanized formal reasoning, *Artificial Intelligence*, **13** (1980), 133-170.