Publ. RIMS, Kyoto Univ.
19 (1983), 169-191

Extracting Lisp Programs from Constructive Proofs: A Formal Theory of Constructive Mathematics Based on Lisp

By

Susumu HAYASHI

There are many attempts to extract programs from formal proofs of theories of constructive mathematics, e. g. [3] [4] [5] [9] [11]. This paper is one of such attempts. The origin of the problem is the so called deductive or theorem-proving approach to the problem of automatic program synthesis. We explain the problem briefly according to [9] and [11]. Let D be a set of data and let $A(x, y)$ be a binary predicate. Then a specification is given by the formula $\forall x \in D \exists y A(x, y)$. A program P is called a solution of the specification, and the predicate A is called the output predicate. The problem of automatic program synthesis is the problem to find a solution automatically from a given specification. A program-synthesis system using the theorem-proving approach finds an appropriate proof of the given specification and extracts a solution from the proof. However, there are no sufficiently powerful theorem-proving systems at the present time. On the other hand, it is relatively easy to construct a solution mechanically from a given proof. Even if the proof is constructed by a person, there is an advantage. If the proof has been checked mechanically, the resulting program does not require debugging or verification.

First we will introduce a formal system called **LM**, which is a modification of $\mathbf{T}_0^{(-)}$ of Feferman [2]. The system $\mathbf{T}_0^{(-)}$ is based on combinatory logic, on the other hand, **LM** is based on a variant of Lisp, which is called Lisp*. Our intended interpretation of the universe of **LM** is the set of S-expressions of Lisp*, on the other hand, the universe of $\mathbf{T}_0^{(-)}$ does not have a fixed intended interpretation. These are the main difference between **LM** and $\mathbf{T}_0^{(-)}$. We will define a universal function of Lisp in **LM**, and using it define a modified q-realizability for **LM**. By the aid of this realizability interpretation, we can extract Lisp programs with formal verifications in **LM** from constructive formal

Communicated by S. Takasu, February 22, 1982.
Institute of Mathematics, The University of Tsukuba, Sakura-mura, Ibaraki, Japan.
Current address: The Metropolitan College of Technology, Asahigaoka 6-6, Hino, Tokyo, Japan.

proofs of $\forall\exists$-theorems in **LM**. However, the version of Lisp used in **LM** is somewhat different from the usual Lisp. Hence the extracted programs are not able to be executed by Lisp interpreters. We will introduce a formal theory **LMI**, and interpret **LMI** in **LM**. Then our realizability interpretation with a slight modification produces programs for the usual Lisp 1.5 interpreters, when it is applied to a proof of **LMI**. One of the defects of this method of implementation is that our interpretation translates proofs of **LMI** into programs with verification proofs of another system **LM**. At any rate, **LM** is a theory-oriented system and the reason why we do not use usual Lisp is to make the system simple from a mathematical point of view.

Let us compare our work with other works [3] [5] [11]. In [3] [4] [11], the normalization method was used, i.e. a term is associated to a proof, and the value of term is calculated through normalizing the term by successive reduction steps. On the other hand, Goto [5] associates terms of primitive recursive functionals to each proof of a specification by the Dialectica interpretation, and associates Lisp programs to the terms. Then the values of terms are calculated by a Lisp interpreter through evaluating the associated programs. However, there are no existing proofs guaranteeing the correctness of the extracted programs as pointed out by Sato [11]. Our fundamental idea is most like to Goto's approach, but we use realizability interpretations. We will also give a proof of the correctness of the extracted programs. The advantages of our method are as follows: Experiences teach us that realizability interpretations are more flexible than normalization methods and more natural than the Dialectica interpretation. Since our realizability interpretation is relatively simple and natural, we can extract programs from semi-formal proofs even by hand. Since realizers are Lisp programs, the extracted programs can be executed by Lisp interpreters. Since our theory is based on Lisp, we can use built-in functions of Lisp systems.

In Section 1 we will introduce **LM**. In Section 2 we will define our realizability interpretation for **LM** and prove the formalized soundness theorem of it. Using the soundness theorem, we will show how to extract Lisp* programs from formal proofs of **LM**. In Section 3 we will introduce **LMI** which is a subsystem of **LM**, and show how to extract Lisp 1.5 programs from proofs of **LMI**. We will extract Wang algorithm of propositional logic as an example.

The author would like to thank Mr. C. Hosono for helpful suggestions and discussions and Dr. M. Sato for his invariable interests.

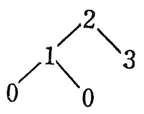
§ 1. A Formal Theory Based on Lisp

In this section we introduce a formal system **LM**, which is a variant of $\mathbf{T}_0^{(-)}$ in Feferman [2]. The system $\mathbf{T}_0^{(-)}$ is based on combinatory logic. On the other hand, our system **LM** is based on a variant of Lisp. First we will explain our Lisp informally. We will denote our Lisp by Lisp*.

The main difference between Lisp and Lisp* is the difference of the data structures. Lisp uses the alphabets, numerals and some special symbols as atoms, and produces all of S-expressions by the dotted pair operation. On the other hand, S-expressions of Lisp* are produced from a single symbol 0 by the dotted pair operation and by the successor operation, which is denoted by $'$. Namely the data structure of Lisp* is defined as follows:

1. 0 is an atom,
2. an atom is an S-expression,
3. if σ is an S-expression, then its successor σ' is an atom.
4. if σ, τ are S-expressions, then the dotted pair $(\sigma.\tau)$ of them is an S-expression.

Note that we may identify an S-expression of Lisp* with a binary tree whose nodes are labelled by non-negative integers. (This was pointed out to the author by Prof. Nobuo Yoneda.) E. g. the S-expression $((0.0)'.0''')''$ is identified with the following tree:



We consider two S-expressions are same iff they are constructed by the same manner. Hence a dotted pair is never an atom. The set of S-expressions is denoted by *Sexp*. A similar but more complicated and sophiscated data structure was adopted in the language HyperLisp (see [11]). The changes of the data structures cause changes of the basic functions. The basic functions of Lisp* are as follows:

1. car, cdr and cons are as usual, i. e.

$$\text{cons}[\sigma;\tau]=(\sigma.\tau),\ \text{car}[(\sigma.\tau)]=\sigma,\ \text{cdr}[(\sigma.\tau)]=\tau,$$

and car, cdr are undefined for atoms.

2. prd, suc are unary functions. The function suc is totally defined and its value is the successor of the argument, and prd is its partially defined inverse function, i. e.

$$\text{suc}[\sigma]=\sigma',\ \text{prd}[\sigma']=\sigma \text{ and prd is undefined for 0 and } (\sigma.\tau).$$

3. zero is a unary function. Its value is either true or false according as its argument is 0 or not.

4. atom is a unary function. Its value is either true or false according as its argument is atom or not.

From these basic functions, we construct programs of Lisp* by McCarthy's conditional forms, λ notations and label notations as usual. We will axiomatize the graph of a universal function *Eval* of *pure* Lisp* in **LM**. By using it we will define a universal function for programs with functional arguments in **LM**. For the mathematical simplicity, we do not axiomatize a universal function for programs with functional arguments. There are some different ways how to treat functional arguments in actual implementations. It will not be reasonable to include axioms which may be changed according to each implementation. Since we are able to define universal functions for each implementation by the aid of *Eval* in **LM**, we can use **LM** as a basic and universal formal system.

We present here a formal description of **LM**.

1. Variables and constants.

1.1. **LM** has only one individual constant 0 (zero).

1.2. Function constants are ′ (successor) and (.) (dotted pair).

1.3. Predicate constants are *Eval*, $=$, *Cl* and $\in$. They are ternary, binary, unary and binary, respectively.

1.4. We use finite strings of *lower case italics* of alphabets and decimal digits whose first character is an alphabet as variables.

We explain the intended meaning of them. The domain of **LM** is regarded as *Sexp*. The equality $=$ is the literal identity between two S-expressions. *Eval* (t_1, t_2, t_3) means that t_1 is a code of a form, t_2 is a code of an a-list and t_3 is the value of t_1 under the environment t_2. Its precise meaning will be determined by the axioms of **LM**.

For the description of the axioms of **LM**, we introduce a system of abbreviations. First we fix a coding of a subset of *Sexp* by finite strings of characters. It is not essential how to code them. It is sufficient to assume that there is an injection from the set of finite strings of *upper case italics* and decimal digits whose first character is an alphabet to *Sexp*. (We include $*$ in alphabets.) We will call these finite strings *Lisp atoms*. Second we introduce M-expressions of Lisp* without functional arguments. These restricted M-expressions are called *PM-expressions*.

1. *car, cdr, cons, suc, prd, zero* and *atom* are *a-constants*. The elements of *Sexp* are *s-constants*. The variables of **LM** whose characters are digits except the first one are *s-variables*. Note that $a, b, c, \cdots$ are s-variables. The variables of **LM** which are not s-variables, a-constants, *quote, label* nor *lambda* are *a-variables*.

2. a-constants and a-variables are *functions* (algorithms). a-constants have the usual arities, and we assume that a-variables have arities in contexts.

3. s-constants and s-variables are forms.

4. If α is a function with arity n and $\sigma_1, \cdots, \sigma_n$ are forms, then $\alpha[\sigma_1; \cdots; \sigma_n]$ is a form.

5. If ε is a form and $x_1, \cdots, x_n$ are s-variables, then $\lambda[[x_1; \cdots; x_n]; \varepsilon]$ is a function of arity n.

6. If α is an a-variable of arity n and β is a form, then label$[\alpha; \beta]$ is a function of arity n.

7. If $\alpha_1, \cdots, \alpha_n, \beta_1, \cdots, \beta_n$ are forms, then $[\alpha_1 \to \beta_1; \cdots; \alpha_n \to \beta_n]$ is a form.

For simplicity we will use the definitions of functions by recursion equations. If such a definition is used, it should be regarded as an abbreviation.

Let i be the fixed injection from the set of Lisp atoms to *Sexp*. Let σ be a variable of **LM**. Let Σ be the expression obtained from σ through replacing all lower cases to upper cases. Let σ^* be $i(\Sigma)$. By the map $\alpha \mapsto \alpha^*$ we define a translation of *PM*-expressions into *S*-expressions as usual. For example, if σ is an s-constant, then σ^* is $(quote^*\ \sigma)$ and $\sigma[\sigma_1; \cdots; \sigma_n]^*$ is $(\sigma^*\ \sigma_1^* \cdots \sigma_n^*)$. Note that we have used the list notation, i.e. $(\alpha_1\ \alpha_2 \cdots \alpha_n)$ is an abbreviation of $(\alpha_1 . (\alpha_2 . (\cdots (\alpha_n . nil^*) \cdots)$. For each natural number n greater than one, *listn* is defined by

$$list\,2[x; y] = cons[x; cons[y; NIL]],$$

$$list\,n{+}1[x_1; x_2; \cdots; x_n] = cons[x_1; list\,n[x_2; \cdots; x_n]].$$

We often write $list[x_1; \cdots; x_n]$ instead of $listn[x; \cdots; x_n]$. For simplicity we abbreviate s-constants $i(*T*)$ and $i(NIL)$ by T and F, respectively, and use atoms of the usual Lisp except T and F to abbreviate the corresponding Lisp* atoms, e.g. NIL means nil^*.

We define *PM*-formulae and their translations into formulae of **LM**. Terms of *PM*-formulae are forms of *PM*-expressions which do not contain free a-variables. Such terms are called *PM*-terms. The atomic *PM*-formulae are $\sigma \simeq \tau$, $\sigma \in \tau$, $Cl(\sigma)$, $\tau\downarrow$, where σ and τ are *PM*-terms. A *PM*-formula is a first order formula constructed from these atomic formulae. We will denote $\exists x(\tau \simeq x\ \&\ Ax)$ by $A(\tau\downarrow)$ and $\neg(\tau\downarrow)$ by $\tau\uparrow$, where x is an s-variable not occurring in the *PM*-term τ. Let F be a *PM*-formula. Then its translation F^* is defined as follows:

1. If τ_2 is not an s-variable, then

$$(\tau_1 \simeq \tau_2)^* \equiv \forall x((\tau_1 \simeq x)^* \longleftrightarrow (\tau_2 \simeq x)^*),$$

where x is an s-variable not occurring in τ_1, τ_2.

2. Let τ be a *PM*-term. Its environment $env(\tau)$ is the following term of **LM**: $((x_1^* . x_1)\ (x_2^* . x_2) \cdots (x_n^* . x_n))$, where $x_1, \cdots, x_n$ is the sequence of free s-variable in τ.

3. $(\tau \simeq x)^*$ is $Eval(\tau^*, env(\tau), x)$, where x is an s-variable.

4. $\tau\downarrow^*$ is $\exists x(\tau \simeq x)^*$, where x is an s-variable not occurring in τ.

5. $(\tau \in \sigma)^*$ is $(\tau\downarrow\ \&\ \sigma\downarrow\ \&\ \forall xy(\tau \simeq x\ \&\ \sigma \simeq y \to x \in y))^*$, where x and y are s-variables not occurring in τ, σ. $Cl(\tau)^*$ is $(\tau\downarrow\ \&\ \forall x(\tau \simeq x \to Cl(x)))^*$. $(x \simeq y)^*$ is $x \simeq y$. $Cl(x)^*$ is $Cl(x)$.

6. $(QxA(x))^*$ is $QxA(x)^*$, $(A \circ B)^*$ is $A^* \circ B^*$ and $(\neg A)^*$ is $\neg A^*$, where Q is a quantifier, $\circ$ is a binary logical connective.

We regard *PM*-formulae as formulae of **LM** by this translation. This translation is a modification of the abbreviations used in [2, II.3]. But $A(\tau)$ means $\exists y(\tau\simeq y\ \&\ A(y))$ in [2].

Let A be a formula of **LM** and let $\bar{x}$ be a sequence of free variables of A. The formula A is elementary with respect to $\bar{x}$ iff it does not contain the predicate symbol Cl and a term t which occurs on the right-hand side of $\in$, i. e. in the context of the form $s\in t$, t is one of the variables of $\bar{x}$. Formulae of $(-)$ type of **LM** are defined inductively as follows:

1. Let A be an atomic *PM*-formula. Then A^* is a formula of $(-)$ type.
2. If A and B are formulae of $(-)$ type, then $A\ \&\ B$, $\forall xA$ are formulae of $(-)$ type.
3. If A is an arbitrary formula of **LM** and B is a formula of $(-)$ type, then $A\rightarrow B$ is a formula of $(-)$ type.

Note that a formula of $(-)$ type may contain existential quantifiers. However, every formula of $(-)$ type whose outermost logical symbol is the existential quantifier has the form $\tau\downarrow^*$.

We use the upper cases for variables ranging over classes. Hence $\forall X(\cdots)$ is $\forall x(Cl(x)\rightarrow\cdots)$ and $\exists X(\cdots)$ is $\exists x(Cl(x)\ \&\ \cdots)$.

By using these abbreviations we describe the axioms of **LM**. For simplicity we denote A^* by A itself. In the following Greek letters indicate *PM*-expressions and $\bar{a}$, $\bar{b}$, $\bar{c}$, $\cdots$ are sequences of variables.

1. The logic of **LM** is the first order Heyting calculus with equality.
2. The axioms on algorithms are as follows.

2.1. $Eval(x, y, z_1)\ \&\ Eval(x, y, z_2)\rightarrow z_1=z_2$.

2.2. $\tau\simeq x\leftrightarrow\tau=x$, where τ is a s-variable or s-constant.

2.3. $cons[a\,;\,b]\simeq x\leftrightarrow x=(a.b)$, $car[0]\uparrow$, $car[a']\uparrow$, $cdr[0]\uparrow$, $cdr[a']\uparrow$, $car[(a.b)]\simeq a$, $cdr[(a.b)]\simeq b$.

2.4. $suc[a]\simeq x\leftrightarrow x=a'$, $prd[a']\simeq a$, $prd[(a.b)]\uparrow$, $prd[0]\uparrow$.

2.5. $suc[a]\not\simeq 0$, $cons[a\,;\,b]\not\simeq 0$, $suc[a]\not\simeq cons[c\,;\,d]$.

2.6. $zero[0]\simeq T$, $zero[a']\simeq F$, $zero[(a.b)]\simeq F$.

2.7. $atom[0]\simeq T$, $atom[a']\simeq T$, $atom[(a.b)]\simeq F$.

2.8. $\lambda[[x]\,;\,\varepsilon][a]\simeq b\leftrightarrow\varepsilon(a/x)\simeq b$, where ε is a *PM*-term and $\varepsilon(a/x)$ is the result of substituting a for x at each occurrence of the free variable in ε.

2.9. $[\alpha_1\rightarrow\beta_1\,;\,\cdots\,;\,\alpha_n\rightarrow\beta_n]\simeq x\leftrightarrow$

$$(\alpha_1\simeq T\ \&\ \beta_1\simeq x)\vee(\alpha_1\simeq F\ \&\ \alpha_2\simeq T\ \&\ \beta_2\simeq x)\vee\cdots\vee$$
$$(\alpha_1\simeq F\ \&\ \alpha_2\simeq F\ \&\cdots\&\ \alpha_{n-1}\simeq F\ \&\ \alpha_n\simeq T\ \&\ \beta_n\simeq x),$$

2.10. $\alpha[\alpha_1\,;\,\cdots\,;\,\alpha_n]\simeq x\leftrightarrow$

$$\exists x_1\cdots x_n(\alpha_1\simeq x_1\ \&\cdots\&\ \alpha_n\simeq x_n\ \&\ \alpha[x_1\,;\,\cdots\,;\,x_n]\simeq x),$$

where $x_1, \cdots, x_n$ are s-variables not occurring in α, $\alpha_1, \cdots, \alpha_n$.

2.11. $label[\alpha; \beta][a_1; \cdots; a_n] \simeq b \leftrightarrow$

$$\beta(label[\alpha; \beta]/\alpha)[a_1, \cdots, a_n] \simeq b.$$

2.12. $\forall a_1 \cdots a_n(\beta(\tau/\alpha)[a_1; \cdots; a_n] \simeq \tau[a_1; \cdots : a_n] \rightarrow$

$$\forall a_1 \cdots a_n(label[\alpha; \beta][a_1; \cdots; a_n]\downarrow \rightarrow \tau[a_1; \cdots; a_n]\downarrow).$$

3. The elementary comprehension axiom ECA$^{(-)}$. We assume that there is a coding of formula of **LM** by S-expressions. We denote the code of F by F^c. We denote $\lambda[[\bar{y}; \bar{Z}]; list[x^c; \bar{y}^c; \bar{Z}^c; F^c; \bar{y}; \bar{Z}]]$ by $\{x; \bar{y}; \bar{Z}|F\}$. Then the following is an axiom scheme:

$$\forall \bar{y}\bar{Z}(Cl(\{x; \bar{y}; \bar{Z}|F\}[\bar{y}; \bar{Z}]) \; \& \; \forall x(x \in \{x; \bar{y}; \bar{Z}|F\}[\bar{y}; Z] \leftrightarrow F),$$

where F is a formula of $(-)$ type and elementary with respect to $\bar{Z}$.

4. Lisp*-induction ***LI****.

$$(A(0) \; \& \; \forall x(A(prd[x]\downarrow) \rightarrow A(x)) \; \&$$
$$\forall x((A(car[x]\downarrow) \; \& \; A(cdr[x]\downarrow) \rightarrow A(x))) \rightarrow \forall x A(x).$$

The final postulate of **LM** is the rule of structural induction. To formulate this we define the class of the natural numbers N. First we define a PM-term *numberp* by

$$numberp[x] = [zero[x] \rightarrow T; \; atom[x] \rightarrow numberp[prd[x]]; \; T \rightarrow F].$$

Let N be the class $\{x \mid numberp[x] \simeq T\}$.

5. The rule of structural induction ***STR***. Let A be a unary formula and let $\prec$ be a binary formula. Define $WF(A, \prec)$ by $\forall x \in N \; A(f(x)) \rightarrow \neg \forall n \in N(f(n') \prec f(n))$, where f is a new function symbol. Let C be a formula of $(-)$ type. If $WF(A, \prec)$ is a theorem of $\mathbf{LM}(f)+C$, then the following is also a theorem of $\mathbf{LM}+C$:

$$\forall x((A(x) \; \& \; \forall y((A(y) \; \& \; y \prec x) \rightarrow B(y))) \rightarrow B(x)) \rightarrow \forall x(A(x) \rightarrow B(x)).$$

This formula will be called the principle of $\prec$-induction and is denoted by $TI(A, \prec, B)$.

For the readers who are not familiar to Feferman's formal theory, we explain the axioms of **LM**. A model of the axioms 2.1-2.12 is called a model of *eval* of Lisp* or simply a model of Lisp* and these axioms are called the axioms of Lisp*. A standard model of Lisp* is a model whose domain is *Sexp* and interpretation of the functions and constants is identical. We assume that any models are standard in this paper. Similarly we consider only standard models of **LM**. We will show that any standard models of Lisp* is expansible to a standard model of **LM**. Hence there will be no troubles, even if we confuse models of **LM** and Lisp*. Even in standard models *Eval* is not necessary to be the minimal solution (fixed point) of the recursion equation suggested by the axioms of Lisp*. On the other hand, $label[\alpha; \beta]$ is the minimal solution of the equation $\alpha = \beta(\alpha)$.

By the aid of the axiom 2.12, we can prove that $label[\alpha;\beta]$ is the minimal solution in each model. Furthermore we can prove in **LM** that any solution represented by a formula of **LM** is larger than $label[\alpha;\beta]$. Hence the axioms of Lisp* determines the truth value of $Eval(\alpha^*, env(\alpha), x)$.

$Cl(a)$ means that a is a class. An S-expression is a class if it codes a suitable subset of *Sexp*. The notation $a\in b$ means that b is a class and a belongs to the set coded by b. It is possible to drop the restriction on the formula F in the axiom ECA$^{(-)}$. However it forces us to use a rather complex realizability interpretation. And we may use classes as input domains of specifications, if we restrict the comprehension to ECA$^{(-)}$. Furthermore the classes generated by the axiom ECA$^{(-)}$ seems to be sufficient to represent data structures. For example any Σ_1^0 sets are classes in our sense. Let $\pi[x;y]$ be a predicate, i.e. a totally defined Lisp* function whose value is either T or F. Let $\varepsilon[n]$ be a function defined on N and enumerating *Sexp*. Define $\alpha[x]$ by $\beta[x;0]$, where $\beta[x;y]=[\pi[x;\varepsilon[y]]\to 0;\ T\to\beta[x;\varepsilon[suc[y]]]]$. Then $\{x\,|\,\alpha[x]\downarrow\}$ is a class and $x\in\{x\,|\,\alpha[x]\downarrow\}$ holds iff $\exists y\pi[x;y]\simeq T$ holds. Hence some finitary inductive definitions of classes are achieved in **LM**. Let X be a new class variable. We define Γ which is the least class of formulae of $\mathcal{L}(\mathbf{LM})(X)$ such that

(i) elementary formulae of **LM** are contained in Γ,

(ii) $\tau\in X$ belongs to Γ, where τ is a *PM*-term,

(iii) if A and B belong to Γ, then $A\&B$, $A\vee B$, $\exists xA$, $\forall x\subseteq\tau A$ belong to Γ, where τ is a *PM*-term not containing x and $x\subseteq y$ means that x is a sub-S-expression of y, i.e. x is obtained from y by successive application of *prd, car, cdr*.

Let $A(X, z)$ be a formula belonging to Γ, then there is a class P such that

(1) $A(P, z)\to z\in P$,

(2) $\forall z(A(Q, z)\to Q(z))\to\forall z(z\in P\to Q(z))$ for each formula Q of **LM**.

This is proved as in [13, § 4]. Next we explain ***SIR***. Assume that $\mathcal{M}$ is a model of $\mathbf{LM}+C$. Assume that $WF(X,\prec)$ is provable in $\mathbf{LM}(f)+C$. Then $\prec$ is a well-founded relation on X in $\mathcal{M}$, since f is a new function symbol. Hence the principle of $\prec$-induction holds in $\mathcal{M}$ by the structural induction of [8]; 5.3.4. Note that we formulate the structural induction not as a scheme but as a rule. It is impossible to formulate it naturally as a scheme without adding the sorts of arbitrary functions.

Let $\mathcal{M}$ be a model of Lisp*. The minimal interpretation of Cl is defined inductively as follows: if a formula A is elementary with respect to $\bar{Z}$ and $\bar{b}$ is a sequence of elements of Cl with the same length as $\bar{Z}$, then Cl contains $\{x;\bar{y};\bar{Z}\,|\,A\}[\bar{a};\bar{b}]$. If b belongs to Cl, then the truth value of $a\in b$ is determined by induction of the definition of Cl. We define that $a\in b$ is false if b does not belong to Cl. Then this expansion is a model of **LM**.

Remark. The structural induction is called the transfinite induction in the area of logic (see [7]) and many formal systems of constructive mathematics are closed under the corresponding derived rule (cf. [6]). The system **LM-*SIR*** will be closed under the rule:

$$\vdash Cl(X) \ \& \ (\forall n\in \boldsymbol{N}(f(n)\in X)\to \exists n\in \boldsymbol{N}\neg(f(n')\prec f(n)))\to \vdash TI(X, \prec, B).$$

§ 2. Realizability Interpretation

In this section we define our realizability interpretation and prove the soundness theorem for it. In the definition of realizability of implication, we need functionals. Hence *PM*-expressions are not sufficient for realizers. We extend *PM*-expressions to *M-expressions* (of Lisp*) by adding functional arguments and define the universal functions of them by *PM*-expressions.

1. *equal* is defined by

$$\begin{aligned}
equal[x\,;\, y]=[&zero[x]\to zero[y]\,;\ atom[x]\to[zero[y]\to F;\\
&\qquad atom[y]\to equal[prd[x]\,;\, prd[y]]\,;\ T\to F]\,;\\
&T\to[atom[y]\to F;\ equal[car[x]\,;\, car[y]]\to\\
&\qquad equal[cdr[x]\,;\, cdr[y]]\,;\ T\to F]].
\end{aligned}$$

2. *caar, cadr, cdar* etc. are defined as usual.
3. *pairlis, assoc* are defined as in [10] 1.6.

From these functions we define the universal functions *evalquote* and *eval* by the following recursion equations. We call this equations E^*.

$$\begin{aligned}
&evalquote[f\,;\, x]=apply[f\,;\, x\,;\, NIL],\\
&apply[f\,;\, x\,;\, a]=[atom[f]\to[equal[f\,;\, CAR]\to caar[x]\,;\\
&\qquad\qquad equal[f\,;\, CDR]\to cdar[x]\,;\\
&\qquad\qquad equal[f\,;\, CONS]\to cons[car[x]\,;\, cadr[x]]\,;\\
&\qquad\qquad equal[f\,;\, ZERO]\to zero[car[x]]\,;\\
&\qquad\qquad equal[f\,;\, ATOM]\to atom[car[x]]\,;\\
&\qquad\qquad equal[f\,;\, SUC]\to suc[car[x]]\,;\\
&\qquad\qquad equal[f\,;\, PRD]\to prd[car[x]]\,;\\
&\qquad\qquad T\to apply[eval[f\,;\, a]\,;\, x\,;\, a]]\,;\\
&\qquad equal[car[f]\,;\, LAMBDA]\to eval[caddr[f]\,;\\
&\qquad\qquad\qquad pairlis[cadr[f]\,;\, x\,;\, a]]\,;\\
&\qquad equal[car[f]\,;\, FUNARG]\to apply[cadr[f]\,;\, x\,;\, caddr[f]]\,;\\
&\qquad equal[car[f]\,;\, LABEL]\to apply[caddr[f]\,;\, x\,;\\
&\qquad\qquad\qquad cons[cons[cadr[f]\,;\, caddr[f]]\,;\, a]]\\
&\qquad T\to apply[eval[f\,;\, a]\,;\, x\,;\, a]],\\
&eval[e\,;\, a]=[atom[e]\to[numberp[e]\to e\,;\, T\to cdr[assoc[e\,;\, a]]]\,;\\
&\qquad equal[car[e]\,;\, QUOTE]\to cadr[e]\,;
\end{aligned}$$

$$\begin{aligned}
&\qquad equal[car[e]; COND] \to evcon[cdr[e]; a];\\
&\qquad equal[car[e]; FUNCTION] \to list3[FUNARG; cadr[e]; a];\\
&\qquad T \to apply[car[e]; evlis[cdr[e]; a]; a]],\\
&evcon[c; a] = [eval[caar[c]; a] \to eval[cadar\ [c]; a];\\
&\qquad\quad T \to evcon[cdr[c]; a]],\\
&evlis[m; a] = [equal[m; NIL] \to NIL;\\
&\qquad\quad T \to cons[eval[car[m]; a]; evlis[cdr[m]; a]]].
\end{aligned}$$

M-expressions processed by these universal functions are defined by the same inductive definition of PM-expressions with the following exceptions:

1. We use *roman letters* instead of italics to write M-expressions. E. g. car, cdr are M-expressions, but *car, cdr* are PM-expressions. We do not distinguish a-variables and s-variables. They are simply called variables. Not only the basic functions and lambda etc., but also function, funarg are not variables.

2. If α is a function, then function$[\alpha]$ is a form. As usual we define a translation from M-expressions into S-expressions. We denote it by σ^+ instead of σ^*. Note that function$[\alpha]^+$ is (FUNCTION α^+). We use M-expressions only as arguments of *evalquote* and *eval* such as $evalquote[\alpha^+; x; y]$, $eval[\sigma^+; x]$. M-expressions were introduced to write Lisp* programs extracted from proofs. On the other hand, PM-expressions are introduced to abbreviate formulae of **LM** by PM-formulae.

There are many sorts of realizability interpretation (see [13]). We adopt q-realizability to extract programs. For each sentence A of **LM**, we associate a formula $\bar{a}\ \boldsymbol{q}\ A$ of **LM**, where $\bar{a}$ is a sequence of variables not occurring in A. $\bar{a}$ may be the empty sequence. The empty sequence is denoted by $\langle\ \rangle$. The formula $\bar{a}\ \boldsymbol{q}\ A$ is called the realizability interpretation of A, $\bar{a}$ are called the realizing variables of A and the length of $\bar{a}$ is denoted by $l(A)$. If $\bar{t}\ \boldsymbol{q}\ A$ holds, then the sequence $\bar{t}$ is called a realizer of A. Realizability interpretation is defined inductively according to the complexity of A. We will abbreviate the formula $\&_{i=1}^{n}\ eval[\alpha_i; x] \simeq y_i$ by $eval[\alpha_1, \cdots, \alpha_n; x] \simeq y_1, \cdots, y_n$ and will use a similar abbreviation for *evalquote.*

Case 1. A is a formula of $(-)$ type. $l(A)=0$. $\langle\rangle\ \boldsymbol{q}\ A$ is A itself.

Case 2. A is not a formula of $(-)$ type.

Case 2.1. A is $A_1 \& A_2$. $l(A)=l(A_1)+l(A_2)$. $\bar{a}, \bar{b}\ \boldsymbol{q}\ A_1\ \&\ A_2$ is

$$\bar{a}\ \boldsymbol{q}\ A_1\ \&\ \bar{b}\ \boldsymbol{q}\ A_2\,.$$

Case 2.2. A is $A_1 \vee A_2$. $l(A)=l(A_1)+l(A_2)+1$. $c, \bar{a}, \bar{b}\ \boldsymbol{q}\ A_1 \vee A_2$ is

$$(c=T\ \&\ A_1\ \&\ \bar{a}\ \boldsymbol{q}\ A_1) \vee (c=F\ \&\ A_2\ \&\ \bar{b}\ \boldsymbol{q}\ A_2)\,.$$

Case 2.3. A is $A_1 \to A_2$. $l(A)=l(A_2)$. $\bar{a}\ \boldsymbol{q}\ A_1 \to A_2$ is

$$\forall \bar{x}((A_1\ \&\ \bar{x}\ \boldsymbol{q}\ A_1) \to \exists \bar{y}(evalquote[\bar{a}; (x_1 \cdots x_n)] \simeq \bar{y}\ \&\ \bar{y}\ \boldsymbol{q}\ A_2)\,,$$

where $\bar{x}$ is $x_1, \cdots, x_n$.

Case 2.4. A is $\forall x Bx$. $l(A)=l(B)$. $\bar{a}\,\boldsymbol{q}\,\forall x Bx$ is

$$\forall x \exists \bar{y}(evalquote[\bar{a}\,;(x)] \simeq \bar{y} \ \& \ \bar{y}\,\boldsymbol{q}\, Bx).$$

The strict meaning of $\bar{y}\,\boldsymbol{q}\,Bx$ is $\bar{y}\,\boldsymbol{q}\,Bc$, where c is the constant of **LM** representing the S-expression x.

Case 2.5. A is $\exists x Bx$. $l(A)=l(B)+1$. $c, \bar{a}\,\boldsymbol{q}\,A$ is

$$Bc \ \& \ \bar{a}\,\boldsymbol{q}\,Bc.$$

We prove the following soundness theorem and obtain Church's rule as a corollary, which enables us to extract Lisp* programs from proofs.

Theorem 1 (soundness theorem). *If A is a provable formula of* **LM** *with the free variables $x_1, \cdots, x_n$, then we can find forms $\alpha_1, \cdots, \alpha_m$ $(m=l(A))$ effectively from a given proof of A such that*

$$\mathbf{LM} \vdash \forall \bar{x} \exists \bar{y}(eval[\alpha_1^+, \cdots, \alpha_m^+;\, e] \simeq \bar{y} \ \& \ \bar{y}\,\boldsymbol{q}\,A),$$

where $\bar{x}$ is $x_1, \cdots, x_n$ and e is $(\mathrm{x}_1^+ .\, x_1) \cdots (\mathrm{x}_n^+ .\, x_n))$.

Proof. The proof is essentially same to the usual soundness theorem of q-realizability interpretation. For readers who are not familiar to realizability interpretation, we will give a precise proof. As logic we use **NJ**. Let Γ be a finite set of formulae of **LM** and let A be a formula of **LM**. Then $\Gamma \Rightarrow A$ is called a sequent. Let Γ be the set $\{A_1, \cdots, A_n\}$, let $p_1, \cdots, p_m$ be the free variables of $\Gamma \cup \{A\}$ and let $\alpha_1, \cdots, \alpha_n$ be a finite sequence of forms whose length is $l(A)$. Then $\Gamma \Rightarrow A : \alpha_1, \cdots, \alpha_n$ denotes the following formula:

$$\&_{i=1}^{n} A_i \ \& \ \bar{a}_i\,\boldsymbol{q}\,A_i \rightarrow \exists \bar{y}(eval[\alpha_1^+, \cdots, \alpha_n^+;\, e] \simeq \bar{y} \ \& \ \bar{y}\,\boldsymbol{q}\,A),$$

where $\bar{a}_i$ is $a_{i1}, \cdots, a_{ip(i)}$ $(p(i)=l(A_i),\ i=1, \cdots, n)$ and e is $((\mathrm{p}_1^+ .\, p_1) \cdots (\mathrm{p}_m^+ .\, p_m)$ $(\mathrm{a}_{11}^+ .\, a_{11}) \cdots (\mathrm{a}_{ij}^+ .\, a_{ij}) \cdots (\mathrm{a}_{np(n)}^+ .\, a_{np(n)}))$. The variables $\bar{a}_i$ are called the realizing variables of A_i. If $\Gamma \Rightarrow A : \bar{t}$ holds, then the sequence $\bar{t}$ is called a realizer of $\Gamma \Rightarrow A$. We prove soundness of the inference rules of **NJ**. In $\exists$***E***-rule and $\forall$***I***-rule, we must change terms of **LM** into forms of M-expressions. Let t be a term of **LM**. Then $\tilde{t}$ is the M-expression obtained from t through replacing $(-.-)$ and $-'$ by cons$[-;\ \ ]$ and suc$[-]$, respectively. Then the inference rules of **NJ** are realizable as follows:

Initial sequent) $\Gamma \Rightarrow A : \bar{a}$ $(A \in \Gamma)$, where $\bar{a}$ are the realizing variables of A.

$$\&\boldsymbol{I})\ \frac{\Gamma_1 \Rightarrow A : \bar{\sigma} \quad \Gamma_2 \Rightarrow B : \bar{\tau}}{\Gamma_1 \cup \Gamma_2 \Rightarrow A\&B : \bar{\sigma}\bar{\tau}}, \qquad \&\boldsymbol{E})\ \frac{\Gamma \Rightarrow A\&B : \bar{\sigma}\bar{\tau}}{\Gamma \Rightarrow A : \bar{\sigma}}, \quad \frac{\Gamma \Rightarrow A\&B : \bar{\sigma}\bar{\tau}}{\Gamma \Rightarrow B : \bar{\tau}},$$

$$\vee\boldsymbol{I})\ \frac{\Gamma \Rightarrow A : \bar{\sigma}}{\Gamma \Rightarrow A \vee B : T, \bar{\sigma}, 0, \cdots, 0}, \qquad \frac{\Gamma \Rightarrow B : \bar{\sigma}}{\Gamma \Rightarrow A \vee B : F, 0, \cdots, 0, \bar{\sigma}},$$

$$\vee E)\quad \frac{\Gamma_1 \Rightarrow A\vee B:\sigma,\bar{\tau}_1,\bar{\tau}_2 \quad \Gamma_2\Rightarrow C:\bar{\alpha} \quad \Gamma_3\Rightarrow C:\bar{\beta}}{\Gamma_1\cup\Gamma_2-\{A\}\cup\Gamma_3-\{B\}\Rightarrow C:\gamma_1,\cdots,\gamma_n},$$

$$\gamma_i=[\sigma\rightarrow\lambda[[\bar{\mathrm{a}}];\alpha_i][\bar{\tau}_1];T\rightarrow\lambda[[\bar{b}];\beta_i][\bar{\tau}_2]] \qquad (i=1,\cdots,n),$$

where $\bar{\alpha}=\alpha_1,\cdots,\alpha_n$, $\bar{\beta}=\beta_1,\cdots,\beta_n$ and $\bar{a}$ and $\bar{b}$ are the realizing variables of A and B, respectively.

$$\rightarrow I)\quad \frac{\Gamma\Rightarrow B:\bar{\sigma}}{\Gamma-\{A\}\Rightarrow A\rightarrow B:\bar{\tau}},\quad \bar{\tau}=\mathrm{function}[\lambda[[\bar{\mathrm{a}}];\sigma_1]],\cdots,\ \mathrm{function}[\lambda[[\bar{\mathrm{a}}];\sigma_n]],$$

where $\bar{a}$ are the realizing variables of A and $\bar{\sigma}=\sigma_1,\cdots,\sigma_n$.

$$\rightarrow E)\quad \frac{\Gamma_1\Rightarrow A\rightarrow B:\bar{\sigma}\quad \Gamma_2\Rightarrow A:\bar{\tau}}{\Gamma_1\cup\Gamma_2\Rightarrow B:\rho_1,\cdots,\rho_m},$$

$$\rho_i=\lambda[[\mathrm{f}i;\mathrm{x1};\cdots;\mathrm{x}n];\mathrm{f}i[\mathrm{x1};\cdots;\mathrm{x}n]][\sigma_i;\bar{\tau}] \qquad (i=1,\cdots,m),$$

where $\bar{\sigma}=\sigma_1,\cdots,\sigma_m$ and $n=l(A)$.

$$\forall I)\quad \frac{\Gamma\Rightarrow Ax:\sigma_1,\cdots,\sigma_n}{\Gamma\Rightarrow\forall xAx:\mathrm{function}[\lambda[[\mathrm{x}];\sigma_1]],\cdots,\mathrm{function}[\lambda[[\mathrm{x}];\sigma_n]]}.$$

$$\forall E)\quad \frac{\Gamma\Rightarrow\forall xAx:\sigma_1,\cdots,\sigma_n}{\Gamma\Rightarrow At:\lambda[[\mathrm{f};\mathrm{x}];\mathrm{f}[\mathrm{x}]][\sigma_1;\tilde{t}],\cdots,\lambda[[\mathrm{f};\mathrm{x}];\mathrm{f}[\mathrm{x}]][\sigma_n;\tilde{t}]}.$$

$\exists I$) If $\exists xAx$ is not a formula of (—) type, then

$$\frac{\Gamma\Rightarrow At:\bar{\sigma}}{\Gamma\Rightarrow\exists xAx:\tilde{t},\bar{\sigma}}.$$

If $\exists xAx$ is a formula of (—) type, then the both realizers of upper and lower sequents are empty.

$\exists E$) If $\exists xAx$ is not a formula of (—) type, then

$$\frac{\Gamma_1\Rightarrow\exists xAx:\sigma,\bar{\tau}\quad \Gamma_2\Rightarrow C:\rho_1,\cdots,\rho_n}{\Gamma_1\cup(\Gamma_2-\{Ax\})\Rightarrow C:\lambda[[\mathrm{x};\bar{a}];\rho_1][\sigma;\bar{\tau}],\cdots,\lambda[[\mathrm{x};\bar{a}];\rho_n][\sigma;\bar{\tau}]},$$

where $\bar{a}=a_1,\cdots,a_m$ are the realizing variables of Ax. If $\exists xAx$ is a formula of (—) type, then it is $(\tau\downarrow)^*$, where τ is a *PM*-expression. We regard τ as an *M*-expression and replace σ by τ in the definition of the realizer of the lower sequent. Then the result is the realizer of the lower sequent in this case.

The lambda notations in the realizers of the lower sequents of the elimination rules are not essential. The realizers of the lower sequents of $\vee E$, $\rightarrow E$, $\forall E$, $\exists E$ may be replaced by $\gamma_i'=[\sigma\rightarrow\alpha_i(\bar{\tau}_1/\bar{a});T\rightarrow\beta_i(\bar{\tau}_2/\bar{b})]$, $\sigma_i[\bar{\tau}]$, $\sigma_i[\tilde{t}]$, $\rho_i(\sigma/x,\bar{\tau}/\bar{a})$, respectively. But they may contradict to the syntax of *M*-expressions in [10]. However, it is easy to see that the Lisp 1.5 interpreter in [10]; Appendix B, does not avoid such extraordinary syntax. Namely we may count $\alpha[\alpha_1;\cdots;\alpha_n]$ as a form, even if α is a form. However this extended syntax causes errors in some actual Lisp systems. It depends on contexts which grammar is more desirable with respect to the problems of size and efficiency. E.g. γ_i is more compact and efficient than γ_i' iff the occurrences of $\bar{a}$ are many. Hence it is rather optional to choose either the traditional syntax or the extended syntax.

All axioms of Lisp* except 2.9, equality axioms and instances of ECA$^{(-)}$ are formulae of (—) type. Hence they are realizable. Note that the usual realizability interpretations of formulae of (—) type are not so simple. A realizer of the axiom 2.9 is given as follows. Let A be $[\alpha_1\to\beta_1;\cdots;\alpha_n\to\beta_n]\simeq x$ and let B be $(\alpha_1\simeq T\ \&\ \beta_1\simeq x)\vee((\alpha_1\simeq F\ \&\ \alpha_2\simeq T\ \&\ \beta_2\simeq x)\vee(\cdots))$. Since $B\to A$ is a formula of (—) type, it is sufficient to realize $A\to B$. It is realizable by the realizer α_1, $[\mathrm{null}[\alpha_1]\to\alpha_2;\ \mathrm{T}\to 0]$, $[\mathrm{null}[\alpha_1]\to[\mathrm{null}[\alpha_2]\to\alpha_3;\ \mathrm{T}\to 0];\ \mathrm{T}\to 0]$, $\cdots$, $[\mathrm{null}[\alpha_1]$ $[\cdots[\mathrm{null}[\alpha_{n-2}]\to[\mathrm{null}[\alpha_{n-1}]\to\alpha_n];\ \mathrm{T}\to 0]\cdots];\ \mathrm{T}\to 0]$. The Lisp* induction ***LI**** is equivalent to the following rule with the eigen variable condition:

$$\textbf{\textit{LIR*}}\qquad \frac{\Gamma_1\Rightarrow A(0)\quad \Gamma_2\Rightarrow A(x')\quad \Gamma_3\Rightarrow A((y\cdot z))}{\Gamma_1\cup\Gamma_2-\{A(x)\}\cup\Gamma_3-\{A(y),A(z)\}\Rightarrow A(u)}\,.$$

Assume that $\Gamma_1\Rightarrow A(0):\alpha_1,\cdots,\alpha_n$, $\Gamma_2\Rightarrow A(x'):\beta_1,\cdots,\beta_n$ and $\Gamma_3\Rightarrow A((y\cdot z))$: $\gamma_1,\cdots,\gamma_n$. Then a realizer of the lower sequent of ***LIR*** is given by $\mathrm{f}_1[\mathrm{u}],\cdots$, $\mathrm{f}_n[\mathrm{u}]$, where

$$\begin{aligned}\mathrm{f}_i[\mathrm{u}]=[&\mathrm{zero}[\mathrm{u}]\to\alpha_i;\\&\mathrm{atom}[\mathrm{u}]\to\lambda[[\mathrm{a}_1;\cdots;\mathrm{a}_n;\mathrm{x}];\beta_i][\mathrm{f}_1[\mathrm{prd}[\mathrm{u}]];\cdots;\mathrm{f}_n[\mathrm{prd}[\mathrm{u}]];\mathrm{prd}[\mathrm{u}]];\\&\mathrm{T}\to\lambda[[\mathrm{b}_1;\cdots;\mathrm{b}_n;\mathrm{c}_1;\cdots;\mathrm{c}_n;\mathrm{y};\mathrm{z}];\gamma_i][\mathrm{f}_1[\mathrm{car}[\mathrm{u}]];\cdots;\\&\qquad\mathrm{f}_n[\mathrm{car}[\mathrm{u}]];\cdots;\mathrm{f}_1[\mathrm{cdr}[\mathrm{u}]];\cdots;\mathrm{f}_n[\mathrm{cdr}[\mathrm{u}]];\mathrm{car}[\mathrm{u}];\mathrm{cdr}[\mathrm{u}]]],\end{aligned}$$

where $a_1,\cdots,a_n$, $b_1,\cdots,b_n$ and $c_1,\cdots,c_n$ are the roman letters corresponding to the realizing variables of $A(x)$, $A(y)$ and $A(z)$, respectively.

Finally we prove that ***SIR*** is sound. Note that $WF(A,\prec)$ is a formula of (—) type. Assume that $\Gamma\Rightarrow WF(A,\prec)$ is provable in **LM**, where Γ is a finite set of (—) type. It is sufficient to realize the sequent $\Gamma\cup\{A(x),Prog\}\Rightarrow B(x)$, where $Prog$ is $\forall x(A(x)\ \&\ \forall y(A(y)\ \&\ y\prec x\to B(y))\to B(x))$. Let $\bar{a}=a_1,\cdots,a_m$ and $\bar{b}=b_1,\cdots,b_n$ be realizing variables of $A(x)$ and $Prog$, respectively. Assume that $\bar{b}\,\boldsymbol{q}\ Prog$ and $Prog$ hold. We define cad^{i}r by car[cdr^i[x]], where cdr^i means the result of $i-1$ times compositions of cdr, e.g. cdr^2[x]=cddr[x]. We define $\tau(\mathrm{x},\bar{\mathrm{a}})=\tau_1(\mathrm{x},\bar{\mathrm{a}}),\cdots,\tau_n(\mathrm{x},\bar{\mathrm{a}})$ as follows:

$$\begin{gathered}\tau_i(\mathrm{x},\bar{\mathrm{a}})=\mathrm{cad}^{i-1}\mathrm{r}[\alpha[\mathrm{x};\bar{\mathrm{a}}]],\quad \alpha=\mathrm{label}[\mathrm{ff};\beta],\quad \beta=\lambda[[\mathrm{x};\bar{\mathrm{a}}]\gamma\delta],\\ \gamma=\lambda[[\mathrm{g}1;\cdots;\mathrm{g}n;\mathrm{x}1;\cdots;\mathrm{x}m+n];\mathrm{list}[\kappa_1;\cdots;\kappa_n]],\\ \delta=[\mathrm{b}_1[x];\cdots;\mathrm{b}_n[x];\mathrm{a}_1;\cdots;\mathrm{a}_m;\lambda_1;\cdots;\lambda_n],\quad \kappa_i=\mathrm{g}i[\mathrm{x}1;\cdots;\mathrm{x}m+n],\\ \lambda_i=\mathrm{function}[\lambda[[\mathrm{x}];\mathrm{function}[\mu_i]]],\quad \mu_i=\lambda[[\bar{\mathrm{a}};\bar{\mathrm{c}}];\mathrm{cad}^{i-1}\mathrm{r}[\mathrm{ff}[\mathrm{x};\bar{\mathrm{a}}]]],\end{gathered}$$

where $\bar{\mathrm{c}}=\mathrm{c}_1,\cdots,\mathrm{c}_p$ and $p=\mathrm{l}(y\prec x)$. Then the sequence $\tau_1(\mathrm{x},\bar{\mathrm{a}}),\cdots,\tau_n(\mathrm{x},\bar{\mathrm{a}})$ is a realizer of $\{A(x),Prag\}\Rightarrow B(x)$. We use the following lemma to prove this.

Lemma 1. *Let a and b be a-lists* $((u_1.v_1)\cdots(u_m.v_m))$ *and* $((w_1.x_1)\cdots(w_n.x_n))$.

Assume that $assoc[u_i; b]$ *has the value* v_i *for* $i=1, \cdots, m$. *Then if* $eval[e; a]$ *has a value, then* $eval[e; b]$ *has the same value. Similarly, if* $apply[fn; arg; a]$ *has a value, then* $apply[fn; arg; b]$ *has the same value.*

This can be proved by the complete computational induction in [9]. Note that we must prove it in **LM**.

Let $F(x)$ be the formula

$$\forall \bar{a}((A(x) \ \& \ \bar{a} \, \mathbf{q} \, A(x)) \rightarrow \exists \bar{y}(eval[\bar{\tau}^+; \varepsilon_0] \simeq \bar{y} \ \& \ \bar{y} \, \mathbf{q} \, B(x)),$$

where ε_0 is $((\mathrm{b}^+.\bar{b})(\mathrm{x}^+.x)\bar{\mathrm{a}}^+.\bar{a}))$. The notation $(\bar{\alpha}^+.\bar{\beta})$ means the sequence $(\alpha_1{}^+.\beta_1) \cdots (\alpha_n{}^+.\beta_n)$, where $\bar{\alpha}$ is $\alpha_1, \cdots, \alpha_n$ and $\bar{\beta}$ is $\beta_1, \cdots, \beta_n$. We will prove that $F(x)$ holds for any x by the aid of $\prec$-induction. Assume that $A(x)$ & $\bar{a} \, q \, A(x)$ holds and $F(y)$ holds for any y such that $y \prec x$. We shall compute $eval[\tau_i(\mathrm{x}, \bar{\mathrm{a}})^+; \varepsilon_0]$.

$$\begin{aligned}
&eval[\tau_i(\mathrm{x}, \bar{\mathrm{a}})^+; \varepsilon_0] \\
&\quad \simeq cad^{i-1}r[eval[\alpha[x; \bar{a}]^+; \varepsilon_0]], \\
&eval[\alpha[\mathrm{x}; \bar{\mathrm{a}}]^+; \varepsilon_0] \\
&\quad \simeq apply[\alpha^+; (x; \bar{a}); \varepsilon_0] \\
&\quad \simeq apply[\beta^+; (x; \bar{a}); \varepsilon_1] \qquad (\varepsilon_1 = (\mathrm{ff}^+.\beta^+).\varepsilon_0)) \\
&\quad \simeq eval[\gamma\delta^+; \varepsilon_2] \qquad (\varepsilon_2 = ((\mathrm{x}^+.x)(\bar{\mathrm{a}}^+.\bar{a}).\varepsilon_1), \\
&\quad \simeq apply[\gamma^+; evlis[\delta^+; \varepsilon_2]; \varepsilon_2].
\end{aligned}$$

Since $\bar{b}$ is a realizer of *Prog*, $evalquote[b_i; (x); NIL]$ has a value. Let v_i be the value. By Lemma 1, we see that

$$v_i \simeq eval[\mathrm{b}_i[\mathrm{x}]^+; \varepsilon_2].$$

Hence $evlis[\delta^+; \varepsilon_2]$ is the list $(v_1 \cdots v_n \ a_1 \cdots a_m \ t_1 \cdots t_n)$, where t_i is (FUNARG $\lambda[[\mathrm{y}]; \mathrm{function}[\mu_i]]^+\varepsilon_2$). Hence we see that

$$\begin{aligned}
&eval[\alpha[\mathrm{x}; \bar{\mathrm{a}}]^+; \varepsilon_0] \\
&\quad \simeq apply[\mathrm{list}^+; (eval[\kappa_1{}^+; \varepsilon_3] \cdots eval[\kappa_n{}^+; \varepsilon_3]); \varepsilon_3],
\end{aligned}$$

where ε_3 is $((\mathrm{g}1^+.v_1) \cdots (\mathrm{g}n^+.v_n)(\mathrm{x}1^+.a_1) \cdots (\mathrm{x}m^+.a_m)(\mathrm{x}m+1^+.t_1) \cdots (\mathrm{x}m+n^+.t_n).\varepsilon_2)$. By the assumption $\bar{a} \, \mathbf{q} \, A(x)$. Assume that $A(y)$ & $y \prec x$ and $\bar{d}, \bar{e} \, \mathbf{q} \, A(y)$ & $y \prec x$ hold.

$$\begin{aligned}
&evalquote[t_i; (y)] \\
&\quad \simeq apply[\lambda[[\mathrm{x}]; \mathrm{function}[\mu_i]]^+; (y); \varepsilon_2] \\
&\quad \simeq (\mathrm{FUNARG} \ \mu_i{}^+((\mathrm{x}^+.y).\varepsilon_2)).
\end{aligned}$$

We denote this value by w_i.

$$\begin{aligned}
&evalquote[\mathrm{w}_i; (\bar{d} \ \bar{e})] \\
&\quad \simeq apply[\mu_i{}^+; (\bar{d} \ \bar{e}); ((\mathrm{x}^+.y).\varepsilon_2))] \\
&\quad \simeq cad^{i-1}r[apply[\beta^+; (y \ \bar{d}); ((\bar{\mathrm{a}}^+.\bar{d})(\mathrm{c}^+.\bar{e})(\mathrm{x}^+.y).\varepsilon_2)]].
\end{aligned}$$

By the induction hypothesis of $\prec$-induction, each $eval[\tau_i{}^+; ((b^+.b)(x^+.y)(\bar{a}^+.\bar{d}))]$ has a value, say u_i, and $u_1, \cdots, u_n \,\boldsymbol{q}\, B(y)$. Note that $cad^{i-1}r[apply[\beta^+; (y\ \bar{d}); ((\mathrm{ff}^+.\beta^+)(\mathrm{b}^+.b)(\mathrm{x}^+.y)(\bar{\mathrm{a}}^+.\bar{d}))]]$ has the value the u_i. By Lemma 1, $evalquote[\mathrm{w}_i; (\bar{d}\ \bar{e})]$ has the same value u_i. We have just proved that $\bar{t}$ is a realizer of $\forall y(A(y)\ \&\ y\prec x \to B(y))$. Note that $\bar{v}$ is a realizer of $(A(x)\ \&\ \forall y(A(y)\ \&\ y\prec x \to B(y)))\to B(x)$. Hence $evalquote[\mathrm{v}_i; (\bar{a}\ \bar{t})]$ has a value, say z_i. By Lemma 1

$$z_i \simeq apply[\mathrm{v}_i; (\bar{a}\ \bar{t}); \varepsilon_3]$$
$$\simeq eavl[\kappa_i{}^+; \varepsilon_3].$$

Hence we see that $eval[\tau_i; \varepsilon_0]\simeq z_i$ and $z_1, \cdots, z_n \,\boldsymbol{q}\, B(x)$ hold. This ends the proof of the soundness theorem.

As a corollary of this theorem, we obtain our main theorem.

Theorem 2. *Let X be a constant of* **LM** *such that* **LM**$\vdash Cl(X)$. *Assume that a sentence $\forall x\in X \exists y A(x, y)$ is provable in* **LM**. *Then there is a form τ (M-expression) such that*

$$\mathbf{LM}\vdash \forall x\in X\exists y(evalquote[\tau^+; (x)]\simeq y\ \&\ A(x, y)).$$

We can find τ effectively from a given proof of $\forall x\in X\exists y A(x, y)$.

Proof. By the soundness theorem, there are forms $\bar{\sigma}=\sigma_1, \cdots, \sigma_n$ $(n=l(\exists y A(x, y))$ such that

$$\mathbf{LM}\vdash x\in X \Rightarrow \exists \bar{y}(eval[\bar{\sigma}^+: ((\mathrm{x}^+.x))]\simeq \bar{y}\ \&\ \bar{y}\,\boldsymbol{q}\,\exists y A(x, y)),$$

since $x\in X$ is a formula of (—) type. Hence we can see that

$$\mathbf{LM}\vdash x\in X \Rightarrow \exists y(eval[\sigma_1{}^+; ((\mathrm{x}^+.x))]\simeq y\ \&\ A(x, y)).$$

Define τ by $\lambda[[\mathrm{x}]; \sigma_1]$.

As was shown in Section 1, we can construct a standard model of **LM**. Let $\mathcal{M}$ be the *a* standard model, and let I be the interpretation function of $\mathcal{M}$ in the sense of [1]. If the formula $\forall x\in X\exists y A(x, y)$ is provable in **LM**, then we can find a form τ effectively such that $\forall x\in X^I\exists y(evalquote^I[\tau^+; (x)]\simeq y\ \&\ A^I(x, y))$, where X^I and A^I etc. are the interpretations of X and A etc. Note that $evalquote^I$ turns out the *minimal* fixed point of the recursion equations E^*. This means that we may regard $evalquote^I$ as an ideal Lisp* processer. Hence τ is a solution of the specification of $\forall x(x\in X^I \to \exists y A^I(x, y))$.

Realizers of a sentence or a sequent are not unique. Hence if one wants to realize a theorem, he should choose effective ones. Let F be a theorem of **LM**, and let P_1 and P_2 be proofs of F. Then realizers $\bar{\alpha}_1$ and $\bar{\alpha}_2$ extracted from P_1 and P_2 by the method of Theorem 1 may be different. Furthermore, there are many possible ways to realize an axiom or an inference rule. We think that various computational structures of a constructive theorem are represented by various realizers of it. A *library of realization* of **LM** is a collection of theorems and rules of **LM** in the following forms:

$$A:\bar{\alpha},\quad \Gamma\Rightarrow A:\bar{\alpha},\quad \frac{\Gamma_1\Rightarrow A_1:\bar{\alpha}_1,\ \cdots,\ \Gamma_n\Rightarrow A_n:\bar{\alpha}_n}{\Gamma\Rightarrow A:\bar{\alpha}}\,.$$

We call them *R-theorem, R-sequent* and *R-rule,* respectively. We regard an *R*-theorem or a *R*-sequent as a sort of routine and an *R*-rule as a sort of linkage program. In the case of proving theorems in a formal system, it is very useful to have enough stores of theorems (lemmata) and derived rules. In the case of writing Lisp programs, sufficient stores of functions are very useful. Similarly it is useful and necessary to have a good library of realization. The following realization of the restricted rule of structural induction $\boldsymbol{SIR}_0$ is one of useful *R*-rules.

$\boldsymbol{SIR}_0$ is a particular case of ***SIR***, but the following realization of it is more efficient that the one given in the proof of Theorem 1.

$\boldsymbol{SIR}_0$

Let Γ be a finite set of formulae of (—) type, let t^i_j ($0\leqq i\leqq n$, $0\leqq j\leqq l_i$) be *PM*-terms and let $C_0, \cdots, C_n, X$ be constants of **LM** such that $Cl(C_i)$, $Cl(X)$ are provable in $\mathbf{LM}+\Gamma$. Assume that the following are provable in $\mathbf{LM}+\Gamma$:

$$x\in C_0,\ x\in X\rightarrow \&^{l0}_{j=0}t^0_j[x]\in X,\ \cdots,\ x\in C_n,\ x\in X\rightarrow \&^{ln}_{j=0}t^n_j[x]\in X\,.$$

Define $y\prec x$ be $\vee^{0\leqq i\leqq n}_{0\leqq j\leqq l_i}(t^i_j[x]\simeq y\ \&\ x\in C_i)$. If $WF(X,\prec)$ is provable in $\mathbf{LM}(f)+\Gamma$, then the following is an *R*-rule of **LM**.

$$\frac{\begin{array}{c}x\in X,\ \Gamma\Rightarrow x\in C_0\vee\cdots\vee x\in C_n:\gamma_0,\ \cdots,\ \gamma_{n-1},\\ \Gamma,\ x\in X,\ x\in C_i,\ A(t^i_0[x]),\ \cdots,\ A(t^i_{l_i}[x])\Rightarrow A(x);\ \sigma^i_0,\ \cdots,\ \sigma^i_m\quad (i=0,\ \cdots,\ m)\end{array}}{\Gamma,\ x\in X\Rightarrow A(x):\tau_0,\ \cdots,\ \tau_m}$$

where $\tau_0, \cdots, \tau_m$ are defined as follows:

$$\begin{aligned}\tau_i[\mathrm{x}]=[&\gamma_0\rightarrow\lambda[[\bar{\mathrm{a}}_0];\ \sigma^0_i][\bar{\tau}[\bar{t}^0[[\mathrm{x}]]];\ \cdots;\\ &\gamma_{n-1}\rightarrow\lambda[[\bar{\mathrm{a}}_{n-1}];\ \sigma^{n-1}_i][\bar{\tau}[\bar{t}^{n-1}[\mathrm{x}]]];\\ &\mathrm{T}\rightarrow\lambda[[\bar{\mathrm{a}}_n];\ \sigma^n_i][\bar{\tau}[\bar{t}^n[\mathrm{x}]]]]\qquad (i=0,\ \cdots,\ m),\end{aligned}$$

$\bar{\mathrm{a}}_i$ is the sequence of variables obtained by concatenating the rearlzing variables of $A(t^i_0), \cdots, A(t^i_1)$,

$$\bar{\tau}[\bar{t}^i[\mathrm{x}]]=b^i_{00},\ \cdots,\ b^i_{0m},\ \cdots,\ b^i_{10},\ \cdots,\ b^i_{1m}\quad (b^i_{pq}\text{ is }\tau_q[t^i_p[x]])\,.$$

Note that this realization does not use functional arguments in opposition to the realization of ***SIR*** given above.

The following instantiation rules $\boldsymbol{ER}_0$, $\boldsymbol{ER}_1$, $\boldsymbol{DR}_0$ and $\boldsymbol{DR}_1$ are also useful *R*-rules. They are *proper R*-rules in the sense that they have no corresponding rules in the language of **LM**.

Let Γ be a finite set of formulae of (—) type.

$$\textbf{ER}_0\ \frac{\Gamma\Rightarrow \exists x A(x):\sigma,\bar{\tau}}{\Gamma\Rightarrow A(eval[\sigma^+;e]):\bar{\tau}},\qquad \textbf{ER}_1\ \frac{\Gamma\Rightarrow \exists x A(x):\sigma,\bar{\tau}}{\Gamma\Rightarrow eval[\sigma^+;e]\downarrow:\langle\rangle}$$

$$\textbf{DR}_0\ \frac{\Gamma\Rightarrow A\vee B:\sigma,\bar{\tau}_1,\bar{\tau}_2}{\{eval[\sigma^+;e]\simeq T\}\cup\Gamma\Rightarrow A:\bar{\tau}_1},\qquad \textbf{DR}_1\ \frac{\Gamma\Rightarrow A\vee B:\sigma,\bar{\tau}_1,\bar{\tau}_2}{\{eval[\sigma^+;e]\simeq F\}\cup\Gamma\Rightarrow B:\bar{\tau}_2},$$

$$\textbf{DR}_3\ \frac{\Gamma\Rightarrow A\vee B:\sigma,\bar{\tau}_1,\bar{\tau}_2}{\Gamma\Rightarrow eval[\sigma^+;e]\in\{T,F\}:\langle\rangle}.$$

In the above rules, e is the environment of σ. Namely, e is the list $((x_1^+.\,x_1)\cdots(x_n^+.\,x_n))$, where $x_1,\cdots,x_n$ are the free variables appearing in σ.

An example of useful R-sequents is

$$\tau\downarrow,\ \tau\not\simeq T\rightarrow\tau\simeq F\Rightarrow\tau\simeq T\vee\tau\simeq F:\tau.$$

Its realizer obtained by the proof of Theorem 1 is a bit different from this one.

§ 3. A Subsystem LMI

We used Lisp* programs and universal functions to realize theorems of **LM**. In this section, we define a system **LMI** and give an interpretation of **LMI** into **LM**. We can regard **LMI** as a subsystem of **LM** by this interpretation. And we will obtain a realizability interpretation of **LMI** by a slight modification of the realizability interpretation in section 2. Since **LMI** is an auxiliary system, we do not give a precise definition of it but give a brief sketch of it. Roughly speaking **LMI** is a system of *PM*-formula based on the usual Lisp.

1. Lisp atoms are constants of **LMI.** We include the numerals of the decimal notation in Lisp atoms. Note that we do not restrict the length of characters of atoms. (In this section, we use roman letters to denote *S*-expressions and *M*-expressions of Lisp.)
2. car, cdr, cons atom, eq, add1, sub1 and numberp are basic function symbols.
3. $\simeq$, $\downarrow$, Cl, $\in$ are atomic predicate symbols.
4. Variables and terms of **LMI** are defined as variables and forms of *M*-expressions.
5. Terms of **LMI** may not have a value. Hence the logic of **LMI** is a logic of partial terms. The following are axioms on partial terms:

$$x\downarrow,\ t\simeq t,\ s\simeq t\leftrightarrow t\simeq s,\ A(s)\ \&\ s\simeq t\rightarrow A(t),\ t\downarrow\leftrightarrow\exists x(t\simeq x),$$
$$t[t_1;\cdots;t_n]\downarrow\rightarrow t_i\downarrow(i=1,\cdots,n),\ Cl(t)\rightarrow t\downarrow,\ s\in t\rightarrow(s\downarrow\&\ t\downarrow),$$

where $s, t, s_1, t_1, \cdots$ are terms.
The axioms and rules on logical connectives and quantifiers are equal to **NJ** except that the following two:

$$\forall xA(x)\ \&\ t\downarrow\rightarrow A(t),\qquad A(t)\ \&\ t\downarrow\rightarrow\exists xA(x).$$

6. Axioms on the basic functions should be defined to express basic properties of such functions. It is rather optional how to define such axioms. Let

A^i be the interpretation of the formula of **LMI** in **LM** which is defined below. Let A be a formula of (—) type. Then A is an axiom of **LMI** iff A^i is a theorem of **LMI**.

7. The axioms 2.8-2.12, $\mathrm{ECA}^{(-)}$ and ***SIR*** are adopted as axioms of **LMI**.
8. ***LI****** is replaced by

$$(\forall x(\mathrm{atom}[x]\simeq T\to A(x))\ \&\ \forall x(A(\mathrm{car}[x]\downarrow)\ \&\ A(\mathrm{cdr}[x]\downarrow)\to A(x)))\to A(t),$$
$$(A(0)\ \&\ \forall x\in N(A(\mathrm{sub1}[x]\downarrow)\to A(x)))\to\forall x\in N\ A(x).$$

The class of natural numbers N is defined by $\{x\mid \mathrm{numberp}[x]\simeq T\}$.

We define an interpretation A^i of a formula A of **LMI** in **LM** as the interpretation of *PM*-formulae in **LM** except the interpretation of atomic formulae. We interpret $(\sigma\simeq\tau)^i$, $(\sigma\downarrow)^i$ etc. as $(\sigma\simeq\tau)^*$, $(\sigma\downarrow)^*$ except using $\mathrm{eval}_0[x;y]\simeq z$ instead of $Eval(x, y, z)$, where eval_0 is the universal functions for Lisp with the basic function of **LMI**. Namely it is the *minimal* solution of the recursion equation E_0 that is obtained from E^* through replacing $[\mathrm{equal}[f;\mathrm{ZERO}]\to\mathrm{zero}[\mathrm{car}[x]]$ etc. by $[\mathrm{equal}[f;\mathrm{NUMBERP}]\to\mathrm{numberp}[\mathrm{car}[x]]$ etc. The functions eq, add1, sub1 are interpreted as follows:

$$\mathrm{eq}[x;y]=[\mathrm{atom}[x]\to[\mathrm{atom}[y]\to\mathrm{equal}[x;y];T\to\nu];T\to\nu],$$
$$\mathrm{add1}[x]=[\mathrm{numberp}[x]\to\mathrm{suc}[x];T\to\nu],$$
$$\mathrm{sub1}[x]=[\mathrm{numberp}[x]\to\mathrm{prd}[x];T\to\nu],$$

where ν is $[0\to 0]$ ($[0\to 0]$ has no value). On the other hand numberp is interpreted by numberp itself. It is obvious that this gives an interpretation of **LMI** in **LM**.

Next we modify the realizability interpretation through replacing evalquote and eval by $\mathrm{evalquote}_0$ and eval_0, respectively. Then we can prove the following theorem as Theorem 2.

Theorem 3. *Let X be a constant of* **LMI** *such that* $\mathbf{LM}\vdash Cl(X)$. *Assume that $\forall x\in X\exists yA(x, y)$ is a theorem of* **LMI**. *Then there is a Lisp program τ built from the basic functions of* **LMI** *such that*

$$\mathbf{LM}\vdash\forall x\in X^i\exists y(\mathrm{evalquote}_0[\tau^*;(x)]\simeq y\ \&\ A^i(x, y)).$$

Let $\mathcal{M}$ be a standard model of **LM**. We may regard $\mathcal{M}$ as an extension of a model of Lisp 1.5, i.e. if $\mathcal{M}\models\mathrm{evalquote}_0[\tau^*;(x)]\simeq y$, then any Lisp 1.5 interpreter computes the same value y. (Of course we neglect the problems of computation time and memory.) Let I be the interpretation function of $\mathcal{M}$. Let A be the set of symbolic atoms of Lisp, let N be the set of numerical atoms and let P be the set of dotted pairs of Lisp. We denote $A\cup P\cup N$ by S. Let $\{A_n\}_{n\in N}$ be a sequence of infinite subsets of A such that $A=\bigcup_{n\in N}A_n$ and $A_m\cap A_n=\emptyset$ for $m\neq n$. Let $\{f_i: A_i\to A_{i-1}\}_{i\in N}$ be a sequence of bijections. We denote the smallest set including a set X and closed under the dotted pair operation by $((X))$. We define a function s as follows:

(i) s is equal to add1 on N,

(ii) $s_{2i+1}: A_{2i+1} \to A_{2i+2}$ is f_{2i+1}. $P_{i+1} = ((A_1 \cup \cdots \cup A_{2i+2} \cup N)) - \bigcup_{j \leq i} P_j$. Choose a bijection $s_{2i+2}: A_{2i+2} \cup P_{i+1} \to A_{2i+3}$.

(iii) Let s be the bijection from S to $S - \{0\}$ induced by $\{s_i\}_{i \in N}$.

Let p be the inverse function of s. Let M_0 be the model of **LM** over S whose interpretation function K is obtained by interpreting suc, prd by s and p, respectively. (The interpretations of the other basic functions are standard.) Set $J = K \circ I \circ i$. Then τ in Theorem 3 gives a solution by Lisp 1.5 programs of the specification $\forall x \in X^J \exists y A^J(x, y)$.

Of course we may realize **LMI** in **LMI**. Hence it is possible to use **LMI** instead of **LM**. However, **LMI** is rather incomplete and optional system. For example **LMI** axioms does not tell us anything about the structures of atoms. On the other hand, we can decompose and compose atoms by the aid of prd and suc in **LM**. Hence, even if we extend **LMI** by adding Lisp 1.5 functions decomposing or composing atoms (without side effects), we can interpret them in **LM**. Furthermore **LMI** is based on the logic of partial terms, on the other hand, **LM** is based on the usual logic. Hence **LM** is more natural and complete than **LMI** from the theoretical point of view.

Now we give an example of extractions by the aid of Theorem 3. In the actual extractions, we may define functions using recursive call instead of the label notations, for we can use actual Lisp interpreters. Furthermore, we may simplify the instantiation rules, through replacing eval[σ^+; e] by σ, since we can prove eval[σ^*; e]$\simeq y \to \sigma \simeq y$ in **LMI** for any Lisp 1.5 programming σ. E. g., we may use the following $\boldsymbol{ER}_0$:

$$\frac{\Gamma \Rightarrow \exists x A(x) : \sigma, \bar{\tau}}{\Gamma \Rightarrow A(\sigma) : \bar{\tau}}.$$

We will adopt the R-rule based on the extended syntax. This will save spaces very much, and the resulting programs do not contradict to the traditional syntax in the following example.

Example. In this example, we extract Wang algorithm of propositional logic. For simplicity we allow only $\neg$ and & as logical connectives and exclude the form $\neg\neg F$ from formulae.

Definition 1. Let *Atom* be the class $\{x \mid \text{atom}[x] \simeq \text{T}\}$. We define the class of formula *Fm* as follows:

(i) $Atom \subseteq Fm$,

(ii) $x \in Fm \to \text{anot}[x] \in Fm$,

(iii) $x, y \in Fm \to \text{sand}[x; y] \in Fm$,

where

anot[x]=[atom[x]$\rightarrow$list[NOT ; x] ;
eq[car[x] ; NOT]$\rightarrow$cadr[x] ; T$\rightarrow$list[NOT ; x]] ,
sand[x ; y]=list[AND ; x ; y] .

Definition 2. A sequent is a list of formulae. We denote the class of sequents by *Seq*. A sequent s is an axiom iff $\exists u(u \in Atom$ & memberp[u ; s] $\simeq$T & memberp[anot[u] ; s]$\simeq$T). We denote the class of axioms by Ax.

Definition 3. We define $P \vdash s$ (P is a proof of s) inductively as follows:
(i) $s \in Ax \rightarrow (0\ \ s) \vdash s$,
(ii) (R1): $\Gamma,\ \neg a,\ \neg b,\ \Delta / \Gamma,\ \neg(a \& b),\ \Delta$. If $P_0 \vdash s_0$ and s_0/s is the rule (R1), then $(1\ P_0\ \ s) \vdash s$.
(iii) (R2): $\Gamma,\ a,\ \Delta ; \Gamma,\ b,\ \Delta / \Gamma,\ a \& b,\ \Delta$. If $P_0 \vdash s_0$, $P_1 \vdash s_1$ and $s_0 ; s_1/s$ is the rule (R2), then $(2\ P_0\ P_1\ \ s) \vdash s$.
Note that $\{(P.\ s) | P \vdash s\}$ is a class. Hence $P \vdash s$ may be regarded a formula of (—) type. It is unnecessary to write down the proof formally. It is possible to write R-proofs (i. e. proof built up from R-theorems and R-rules) semiformally.

We give a semi-formal R-proof of the following proposition and extract Wang algorithm from its proof.

Proposition. $s \in Seq \Rightarrow \exists P(P \vdash s) \vee \nvdash s$, *where* $\nvdash s$ *is an abbreviation of* $\neg \exists P(P \vdash s)$.

Assume that this sequent is provable and σ, τ is its realizer. Then we can decide either that s is provable or not by evaluating σ. And if the value is T, then the value of τ gives a proof of s.
We need some lemmata to prove the proposition.

Lemma 1. *Set*

$$A(s) \equiv \forall x((\text{memberp}[x\ ;\ s] \simeq \text{T}\ \&\ x \notin Atom) \rightarrow (\text{car}[x] \simeq \text{NOT}\ \&\ \text{cdr}[x] \notin Atom)),$$

$$B(s) \equiv \exists xyzB_0,\ C(s) \equiv \exists xyzC_0,$$

$$B_0 \equiv (x,\ z \in Seq\ \&\ y \in Fm\ \&\ \text{car}[y] \simeq \text{NOT}\ \&\ \text{cadr}[y] \simeq \text{AND}\ \&\ \text{app3}[x\ ;\ y\ ;\ z] \simeq s),$$

$$C_0 \equiv (x,\ z \in Seq\ \&\ y \in Fm\ \&\ \text{car}[y] \simeq \text{AND}\ \&\ \text{app3}[x\ ;\ y\ ;\ z] \simeq s),$$

where app3[x ; y ; z]=append[x ; cons[y ; z]]. Then the sequent $s \in Seq \Rightarrow A(s) \vee (B(s) \vee C(s))$ holds.

Proof. We denote $A(s) \vee B(s) \vee C(s)$ by $F(s)$. Assume that $s \in Seq$. Then null[s]$\simeq$T or not.
1. null[s]$\simeq$T. Then $A(s)$ holds. Hence $F(s)$: T, 0, $\cdots$, 0.
2. Otherwise. Then car[s]$\in Fm$ and cdr[s]$\in Seq$. Assume that F((cdr[s]) : $c1, \cdots, c8$. We consider the cases car[s]$\in Atom$ or not.

2.1. atom[car[s]]$\simeq$T. If A(cdr[s]), then $F(s)$: T, 0, $\cdots$, 0. If B(cdr[s])$\vee$ C(cdr[s]): $c2, \cdots, c8$, then $F(s)$: F, $c2, \cdots, c8$.

2.2. Otherwise. Then caar[s]$\in$*Atom*. We consider the cases caar[s]$\simeq$NOT or not.

2.2.1. eq[caar[s]; NOT]$\simeq$T. Then cadar[s]$\in$*Fm*. We consider the cases cadar[s] is an atom or not.

2.2.1.1. atom[cadar[s]]$\simeq$T. If A(cdr[s]), then $F(s)$: T, 0, $\cdots$, 0. If B(cdr[s]) $\vee C$(cdr[s]): $c2, \cdots, c8$, then $F(s)$: F, $c2, \cdots, c8$.

2.2.1.2. Otherwise. Then caadar[s] is AND. Hence we see that $F(s)$: F, T, NIL, car[s], cdr[s], 0, 0, 0.

2.2.2. eq[caar[s]; NOT]$\simeq$F. Then caar[s] is AND. Hence we see that $F(s)$: F, F, 0, 0, 0, NIL, car[s], cdr[s].

Hence an realizer f1, $\cdots$, f8 of the lemma is given by the R-rule of **SIR**$_0$ as follows:

$$\text{f1}[s]=[\text{null}[s]\to\text{T};\ \text{T}\to\gamma_1(\text{f1}[\text{cdr}[s]]/c1],$$
$$\text{f}i[\text{s}]=[\text{null}[\text{s}]\to 0;\ \text{T}\to\gamma_i(\text{f}i[\text{cdr}[s]]/ci)] \qquad (i=2, \cdots, 8).$$
$$\gamma_1=\gamma(\text{T, F, F, F}),\quad \gamma_2=\gamma(0, c2, \text{T, F}),\quad \gamma_3=\gamma(0, c3, \text{NIL}, 0),$$
$$\gamma_4=\gamma(0, c4, \text{car}[s], 0),\quad \gamma_5=\gamma(0, c5, \text{cdr}[s], 0),\quad \gamma_6=\gamma(0, c6, 0, \text{NIL}),$$
$$\gamma_7=\gamma(0, c7, 0, \text{car}[s]),\quad \gamma_8=\gamma(0, c8, 0, \text{cdr}[s]),$$

where $\gamma(x0, x1, x2, x3)$ is

$$\begin{aligned}&[\text{atom}[\text{car}[s]]\to[c1\to x0;\ \text{T}\to x1];\\ &\quad \text{T}\to[\text{eq}[\text{caar}[s];\ \text{NOT}]\to[\text{atom}[\text{cadar}[s]]\to\\ &\qquad x[c1\to x0;\ \text{T}\to x1];\ \text{T}\to x2];\ \text{T}\to x3]].\end{aligned}$$

Lemma 2. $s\in Seq,\ A(s)\Rightarrow s\in Ax\vee s\notin Ax.$

Proof. This is proved by using ***SIR***$_0$ on the structure of *Seq*. Let $G(s)$ be the formula $s\in Ax\vee s\notin Ax$.

1. null[s]$\simeq$T. Then $G(s)$: F.

2. null[s]$\simeq$F. Then cdr[s]$\in$*Seq* and A(cdr[s]). Assume that G(cdr[s]): ax[cdr[s]].

2.1. cdr[s]$\in Ax$. Then $G(s)$: T.

2.2. cdr[s]$\notin Ax$.

2.2.1. memberp[anot[car[s]]; cdr[s]]$\simeq$T. Then $G(s)$: T.

2.2.2. Otherwise. Then $G(s)$: F.

Hence a realizer ax[s] of the lemma is given by

$$\begin{aligned}&\text{ax}[s]=[\text{null}[s]\to\text{F};\ \text{T}\to[\text{ax}[\text{cdr}[s]]\to\text{T};\\ &\qquad \text{T}\to[\text{memberp}[\text{anot}[\text{car}[s]];\ \text{cdr}[s]]\to\text{T};\ \text{T}\to\text{F}]]].\end{aligned}$$

By these twe R-lemmata, we obtain an R-proof of the proposition by ***SIR***$_0$. Let $H(s)$ be $\exists P(P\vdash s)\vee\not\models s$. By lemmata 1, 2, the following are R-sequents:

$$s\in Seq \Rightarrow s\in C_1\vee s\in C_2\vee s\in C_3: \mathrm{f1}[s],\ \mathrm{f2}[s],$$
$$s\in Seq,\ s\in C_1 \Rightarrow s\notin Ax\vee s\Subset Ax: ax[s],$$
$$s\in Seq,\ s\in C_2 \Rightarrow B_0(\mathrm{f3}[s],\ \mathrm{f4}[s],\ \mathrm{f5}[s]): \langle\ \rangle,$$
$$s\in Seq,\ s\in C_3 \Rightarrow C_0(\mathrm{f6}[s],\ \mathrm{f7}[s],\ \mathrm{f8}[s]): \langle\ \rangle.$$

where C_1, C_2, C_3 are the classes $\{s|\mathrm{f1}[s]\simeq\mathrm{T}\}$, $\{s|\mathrm{f1}[s]\simeq\mathrm{F}\ \&\ \mathrm{f2}[s]\simeq\mathrm{T}\}$, $\{s|\mathrm{f1}[s]\simeq\mathrm{f2}[s]\simeq\mathrm{F}\}$, respectively. Define rn, ra1, ra2 as follows:

$$\mathrm{rn}[s]=\mathrm{app4}[\mathrm{f3}[s];\ \mathrm{cadadr}[\mathrm{f4}[s]];\ \mathrm{caddadr}[\mathrm{f4}[s]];\ \mathrm{f5}[s]],$$
$$\mathrm{ra1}[s]=\mathrm{app2}[\mathrm{f6}[s];\ \mathrm{cadr}[\mathrm{f7}[s]];\ \mathrm{f8}[s]],$$
$$\mathrm{ra2}[s]=\mathrm{app2}[\mathrm{f6}[s];\ \mathrm{caddr}[\mathrm{f7}[s]];\ \mathrm{f8}[s]],$$

where

$$\mathrm{app4}[x;\ y;\ u;\ v]=\mathrm{append}[x;\ \mathrm{cons}[y;\ \mathrm{cons}[u;\ v]]].$$

Then we see that $s\in\mathrm{Seq}$, $s\in C_2\Rightarrow \mathrm{rn}[s]\in Sep$,

$s\in\mathrm{Seq}$, $s\in C_3\Rightarrow \mathrm{ra1}[s]$, $\mathrm{ra2}[s]\in Sep$.

The functions rn, ra1, ra2 decrease the number of conjunctions. Assume that $s\in Seq$ and $s\in C_1$. If $s\in Ax$, then $H(s)$: T, list[0; s]. Otherwise $H(s)$: F, 0. Hence $s\in Seq$, $s\in C_1\Rightarrow H(s)$: s00[s], s10[s], where s00[s]=[ax[s]→T; T→F], s10[s]=[ax[s]→list[0; s]; T→0].

Assume that $s\in\mathrm{Seq}$, $s\in C_2$ and $H(\mathrm{rn}[s])$: a_0, a_1.

1. $\exists P(P\vdash\mathrm{rn}[s])$: a_1. Then $a_1\vdash\mathrm{rn}[s]$. Hence $H(s)$: T, list[1; a_1; s].

2. $\nvdash\mathrm{rn}[s]$. Then $\nvdash s$. (Since $\nvdash s$ is a formula of (—) type, it is out of problem how to prove it.) Hence $H(s)$: F, 0.

Hence we see that

$$s\in Seq,\ s\in C_2,\ H(\mathrm{rn}[s])\Rightarrow H(s): \mathrm{s01}[a_0;\ s],\ \mathrm{s11}[a_0;\ a_1;\ s],$$
$$\mathrm{s01}[a_0;\ s]=[a_0\to\mathrm{T};\ \mathrm{T}\to\mathrm{F}],\ \mathrm{s11}[a_0;\ a_1;\ s]=[a_0\to\mathrm{list}[1;\ a_1;\ s];\ \mathrm{T}\to 0].$$

Let a_2, a_3 and a_4, a_5 be realizing variables of $H(\mathrm{ra1}[s])$ and $H(\mathrm{ra2}[s])$, respectively. Then we can prove that

$$s\in Seq,\ s\in C_3,\ H(\mathrm{ra1}[s]),\ H(\mathrm{ra2}[s])\Rightarrow H(s): \mathrm{s02}[a_2;\ a_4;\ s],\ \mathrm{s12}[a_2;\ \cdots;\ a_5;\ s],$$

where

$$\mathrm{s02}[a_2;\ a_4;\ s]=[a_2\to[a_4\to\mathrm{T};\ \mathrm{T}\to\mathrm{F}];\ \mathrm{T}\to\mathrm{F}],$$
$$\mathrm{s12}[a_2;\ \cdots;\ a_5;\ s]=[a_2\to[a_4\to\mathrm{list}[2;\ a_3;\ a_5;\ s];\ \mathrm{T}\to 0];\ \mathrm{T}\to 0].$$

By $\mathbf{SIR}_0$ we see that

$$s\in Seq\Rightarrow H(s): \mathrm{r0}[s],\ \mathrm{r1}[s],$$
$$\mathrm{r0}[s]=[\mathrm{f1}[s]\to\mathrm{s00}[s];\ f2[s]\to\mathrm{s01}[\mathrm{r0}[\mathrm{rn}[s]];\ s];$$
$$T\to\mathrm{s02}[\mathrm{r0}[\mathrm{ra1}[s]];\ \mathrm{r0}[\mathrm{ra2}[s]];\ s]],$$
$$\mathrm{r1}[s]=[\mathrm{f1}[s]\to\mathrm{s10}[s];\ f2[s]\to\mathrm{s11}[\mathrm{r0}[\mathrm{rn}[s]];\ \mathrm{r1}[\mathrm{rn}[s]];\ s];$$
$$T\to\mathrm{s12}[\mathrm{r0}[\mathrm{ra1}[s]];\ \mathrm{r1}[\mathrm{ra1}[s]];\ \mathrm{r0}[\mathrm{ra2}[s]];\ \mathrm{r1}[\mathrm{ra2}[s]]]].$$

References

[1] Chang, C. C. and Keisler, H. J., *Model Theory*, North-Holland, Amsterdam London, 1973.

[2] Feferman, S., Constructive theories of functions and classes, *Logic Colloq.* **78**, D. van Dalen, M. Boffa and K. MacAloon, eds., (1978), 159-224.

[3] Goad, C. A., Proofs as descriptions of computation, *Lecture Notes in Computer Science*, **87** (1980), 39-52.

[4] ———, Computational uses of the manipulation of formal proofs, *Ph. D Thesis, Stanford University*, 1980.

[5] Goto, S., Program synthesis from natural deduction proofs. *Proc. IJCAI, Tokyo* (1979), 339-341.

[6] Hayashi, S., A note on the bar induction rule, *The L.E.J. Brouwer Centenary Symposium*, A.S. Troelstra, D. van Dalen, eds., (1982), 149-163.

[7] Kreisel, G. and Howard, W., Transfinite induction and bar induction of types zero and one, and the role of continuity in intuitionistic analysis, *The Journal of Symbolic Logic*, **31** (1966), 325-358.

[8] Manna, Z., *Mathematical Theory of Computation*, McGraw-Hill, New-York, 1974.

[9] Manna, Z. and Waldinger, R., Toward automatic program synthesis, *Communications of ACM*, **14** (1971), 151-165.

[10] McCarthy, J. et al., *LISP 1.5 Programmer's Manual*, MIT Press, 1962.

[11] Sato, M., Towards a mathematical theory of program synthesis, *Proc. IJCAI*, Tokyo, (1979), 757-762.

[12] ———, Theory of symbolic expressions, I, *Technical Report* **81-13**, Department of Information Science, University of Tokyo.

[13] Troelstra, A. N., Metamathematical investigations of intuitionistic arithmetic and analysis, *Lecture Notes in Mathematics*, **344** (1973).