ICOT Technical Report: TR-224

TR-224

AN APPROACH TO PROOF CHECKER

by

K. HIROSE (Waseda Univ.)

K. YOKOTA and K. SAKAI

January, 1986

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

AN APPROACH TO PROOF CHECKER

Ken HIROSE (Department of Mathematics, Waseda University)
Kazumasa YOKOTA (4th Laboratory, ICOT)
Ko SAKAI (2nd Laboratory, ICOT)

1. INTRODUCTION

Automated theorem proving or automated deduction underwent a quarter of a century of research and development and today it forms one of the oldest areas in artificial intelligence (AI). Meanwhile there have been implemented many interesting theorem provers and proof checkers. It is recognized however that most of them are devoted to deal with only "formal" proofs and that there have been developed rather a few proof checkers. Here, proof checker denotes a system which checks the correctness of given proofs. Moreover, there seems to be good reason now to make a study of not only proof checking but also formal proof generation, in view of status quo of automated theorem proving. Nevertheless, when proofs to be checked are not stated "formally" but informally, their proof checker should be powerful enough to check them, where the "informality" induces different kinds of problems not encountered in the systems dedicated to formal proofs, and hence deserves a harder investigation yet to do. A proof checker for informally stated proofs has to fill possible gaps of inference occurring between proof steps within a given informal proof. It is very often that these gaps are very wide. For instance, if a given proof consists of only a conclusion, the proof checker would have virtually the same capability of generating its proof with a completely automated theorem prover. In addition, the proof checker should have some facilities for acquiring or learning knowledge of proofs and theorems to achieve amicable and efficient proof checking. Therefore, a proof checker for informal proofs should have many functions such as theorem proving, knowledge base for mathematical theories, and certain inductive inference, or inductive learning, based on the knowledge base in addition to deductive inference functions. Hence, it is extremely difficult to construct a practical and powerful theorem prover or proof checker, where the word "practical" means to handle informal proofs. Continuous efforts will be requested to gradually resolve the difficulties impeding us from making it a good assistant or an efficient aid for education.

In my opinion, the perspectives of knowledge information processing will provide a powerful approach to such a proof checker, since it will be considered as an inference and computation (or ratiocination) system coupled with a certain knowledge base. I think this approach would play a fundamental role in the following points:

(1) It will give a cue to a formalization of inductive theory,
(2) it will be able to clarify relations between various propositions and

This paper is based on an invited talk by K.Hirose at the Twelfth International Symposium of Mathematical Foundations of Computer Science'86 (MFCF'86), Bratislava, Czechoslovakia, August 25-29, 1986.

methods,

(3) it will also help to clarify mutual relations between mathematical theories, and

(4) it will provide a new approach to metamathematical treatment of mathematics, that is, a metatheory not confined to a single particular mathematical theory, but a theory about multiple correlated mathematical theories.

Knowledge information processing, in other words, the construction of an ideal intelligent man-machine system is one of the major goals at ICOT (Institute for New Generation Computer Technology). Since the above mentioned proof checker is considered as an archetype AI system, a research project on proof checking system has been established as one of R&D goals at ICOT under the title of the Computer Aided Proof (CAP) project. This paper describes the status of ICOT's CAP project.

2. OVERVIEW OF CAP PROJECT

The Fifth Generation Computer System (FGCS) Project in Japan is a ten-year program (1982-1991) being pursued by ICOT and its Working Groups. At its initial stage (1982-84), many tools and systems were developed for use in R&D activities at the intermediate stage (1985-1988). The CAP system is situated in a framework of an intelligent programming (IPS) project, which is one of the important projects of FGCS. In brief, IPS is a system for symbolic computation which processes specifications, programs, logical formulas as well as algebraic formulas. It uses theorem proving or proof checking techniques for program derivation (transformation), program verification, term rewriting, formula manipulation and so on. Thus, CAP plays a central role in IPS.

CAP has gone into effect in 1983 as a subproject of ICOT Working Group 5, and many theoretical issues were discussed [1,2] there. It will be active until the end of FGCS project, i.e., 1991. The final goal of this project is to construct a general proof checker incorporating a large amount of knowledge common to working mathematicians, with various utilities such as proof editor, pretty printer (with two-dimensional display), and symbolic manipulator of mathematical formulas. Because of the restricted resources available for us and other limitations imposed on us, we initiated the project by tentatively building a simple proof checker for some concrete branches of mathematics and then by successively upgrading it into a general proof checker, instead of directly undertaking the construction of a general one.

As a matter of fact, it took a lot of effort to implement even a simple proof checker. A formal language had to be designed for use in writing theorems and proofs, and proof editor for mathematical text with complex structures, proof checking facilities and knowledge base management facilities to store and control many definitions, theorems and proofs, and to maintain their consistency were also required. Many experimentations are now underway on computer-assistance for solving mathematical problems, for the purpose of getting an ideal interface for man-machine collaboration on such activities in particular. We also plan to enhance the system into a central subsystem of IPS, which facilitates such functions as program derivation and verification.

In the first place, we selected three branches of mathematics as the target of our proof checker.

(1) Linear Algebra (LA)
(2) QJ
(3) Synthetic differential geometry (SDG)

Linear algebra is the most familiar branch of college mathematics. We selected the textbook by S. Furuya "Matrices and Determinants" for a freshman cource and designed a formal language for writing all theorems and their proofs in the book. Another reason for selecting linear algebra was to perform an experiment with a two-dimensional display. This proof checking system is named CAP-LA and based on Gentzen's natural deduction system (NK) with some additional inference rules.

The second one, QJ, is a formal system developed by Prof. M. Sato of Tohoku University [3], intended as both a logical system and a programming system. QJ is a constructive system based on free intuitionistic logic. The programming language Quty (formerly called Qute [4]), which itself is based on the system QJ, will be an implementation language of the QJ proof checker. Self-referential expressions can also be written in QJ. Hence we can write metatheorems in it. The first important target of this proof checker, called CAP-QJ, is to check the incompleteness theorem.

The third one, synthetic differential geometry, is a very new field of mathematics originated by A. Kock [5]. The theory aims to study differential geometry in a synthetic manner by introducing infinitesimal objects. The proof checking system, called CAP-SDG [6], will be based on a interactively controlled term rewriting system to check a proof line by line.

The implementation of CAP-LA is running ahead of the others in CAP project. In what follows I will mainly describe the CAP-LA system.

3. CAP-LA SYSTEM

The first version of CAP-LA system was implemented on the PSI machine by a programming language called Extended Self-contained Prolog (ESP). PSI is the personal sequential inference machine developed by ICOT during three years at the initial stage and is now being used as an R&D tool for many FGCS knowledge processing systems [7,8]. ESP is a logic programming language with an "object" concept, designed for the PSI machine [9]. The operating and programming system of the PSI is also written in ESP.

Proof Description Language (PDL, the first version) was designed for the CAP-LA system and over 90 % of theorems and their proofs in the above book have been written in it. The first version of CAP-LA consists of the following four main modules:

(1) System Controller,
(2) Proof Editor,
(3) Proof Checker,
(4) Knowledge Base Manager.

Their configuration is shown in Figure 1.

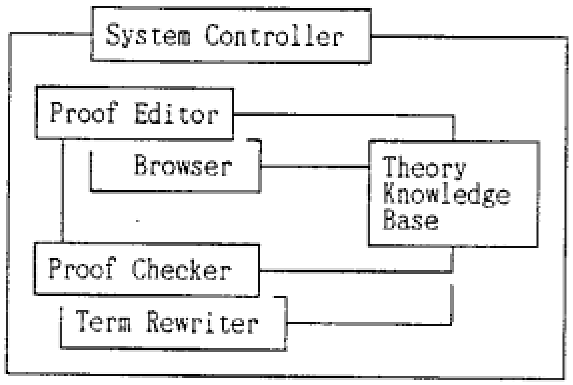


Fig.1 Configuration of CAP-LA

The system controller controls the other modules of the system and provides a man-machine interface. The proof editor is a structured editor dedicated to mathematical proof and has special knowledge about the syntax of the Proof Description Language (PDL). The editor includes a browser editor. The proof checker is a kernel module of the system. It controls general knowledge of logical inference and special knowledge of linear algebra and checks whether a given proof is correct or not. The checker also contains a term rewriting system to check equality of terms. The knowledge base manager contains various kinds of systematically organized knowledge such as definitions, theorems and proofs (checked or not) in the form of terms. It controls their consistency, and retrieves necessary information by unification during proof checking. Upon completion of checking, it inserts new inference rules for successive proof steps.

The implementation of the first version of CAP-LA system was completed in March, 1986 and tests are now underway. The system consists of 4,000 lines of ESP and four programmers working for six months was required for its implementation.

4. PROOF DESCRIPTION LANGUAGE

PDL plays a central role in man-machine interaction. PDL should facilitate reading and writing every mathematical proof and at the sametime PDL should suffice the requirement that every written proof in it be convertible into machine-readable format for further use in proof checking. Moreover, PDL should have rich expressive power enough to represent various kinds of mathematical proofs. We selected Gentzen's natural deduction system (NK) as its basis, since it seems to reflect intrinsically the process of human introduction rule

$$\frac{\overset{\vdots}{A}}{A \vee B} \qquad \frac{\overset{\vdots}{A} \quad \overset{\vdots}{B}}{A \wedge B} \qquad \frac{\overset{[a:t]}{\overset{\vdots}{A(a)}}}{\forall x{:}t.A(x)}$$

elimination rule

$$\frac{\overset{\vdots}{A \vee B} \quad \overset{[A]}{\overset{\vdots}{C}} \quad \overset{[B]}{\overset{\vdots}{C}}}{C} \qquad \frac{A \wedge B}{A} \qquad \frac{\forall x{:}t.A(x) \quad a{:}t}{A(a)}$$

('a:t' means that 'a is an element of type t')

Fig.2 Inference Rule Schemas of NK

```
∨-introduction rule
                        :
                        A
                        hence A ¦ B
∧-introduction rule
                        :
                        A
                        :
                        B
                        hence A & B
∀-introduction rule
                       all x:t.A(x)
                         since
                           let a:t be arbitrary
                             :
                           A(a)
                         end_since
∨-elimination rule
                       A | B
                        hence C
                         since divede and conquer A ¦ B
                           case A
                             :
                             C
                           case B
                             :
                             C
                         end_since
∧-elimination rule
                           :
                        A & B
                        hence A
∀-elimination rule
                        all x:t.A(x)
                        a:t
                        A(a)
```

Fig.3 Corresponding PDL Proof Templates

interface and we preferred to have logical completeness of our system. Figure 2 depicts two inference rules, an introduction rule and an elimination rule of NK. And Figure 3 shows the proof templates in PDL syntax corresponding to rules of NK in Figure 2.

Similarly, other inference rules of NK also correspond to the proof templates in PDL. Many inference rules are necessary to develop mathematical theories, and they also correspond to the proof templates in PDL (Fig.4,5). We built approximately fifty rules into the first version of CAP-LA. Once a proof of a theorem is checked and found to be correct, it is registered into the system knowledge base as an inference rule for further use in the successive proof checking environment.

$$\frac{\begin{array}{c}\vdots\\ A(0)\end{array}\qquad\qquad\begin{array}{c}[n{:}nat]\ [A(n)]\\ \vdots\qquad\vdots\\ A(n+1)\end{array}}{A(n)}$$

Fig.4 Mathematical Induction

```
all m:nat.A(m)
since induction on m
  base
    :
    A(0)
  step
    let n:nat be such that A(n)
    :
    A(n+1)
end_since
```

Fig.5 Mathematical Induction in PDL

The basic policies underlining the first version of PDL are as follows:

(1) The syntax of PDL includes the proof templates corresponding to the inference rules of NK.
(2) It should be a strongly typed system in which all elements are typed. The user should be able to define new types with parameters. Types are handled in the same way as other logical formulas.
(3) Constants (such as integers and 0-vector), function symbols, and variables are not distinguished syntactically. All the bound variables must be bound explicitly by a quantifier. The user should be able to define new constants and function symbols. Constants and free variables are logically handled in the same way.
(4) The user should be able to use naturally in a proof conjunctions, adjectives and adverbs.

(5) The user should be able to specify the inference to be applied at each proof step or to skip checking.
(6) The system contains some special knowledge about linear algebra. Vectors are to be considered as one-column or one-row matrices.

The syntax of PDL will not be detailed, since it is very easy to understand. It will be delineated by making use of a simple theorem and its proof. Figure 6 shows a simple theorem on the determinant of transpose and its proof quoted from "Linear Algebra" by S. Lang, and Figure 7 shows the corresponding description in PDL.

<u>Theorem 6</u>. Let A be a square matrix. Then $\mathrm{Det}(A) = \mathrm{Det}({}^tA)$.

<u>Proof</u>. In Theorem 5, we had
(*) $\mathrm{Det}(A) = \sum_\sigma \varepsilon(\sigma)\, a_{\sigma(1),1} \cdots a_{\sigma(n),n}$.
Let σ be a permutation of $\{1,\cdots,n\}$.
If $\sigma(j) = k$, then $\sigma^{-1}(k) = j$.
We can therfore write
$$a_{\sigma(j),j} = a_{k,\sigma^{-1}(k)}.$$
In a product
$$a_{\sigma(1),1} \cdots a_{\sigma(n),n}$$
each integer k from 1 to n occurs precisely once among the integers $\sigma(1),\cdots,\sigma(n)$. Hence this product can be written
$$a_{1,\sigma^{-1}(1)} \cdots a_{n,\sigma^{-1}(n)}$$
and our sum (*) is equal to
$$\sum_\sigma \varepsilon(\sigma^{-1})a_{1,\sigma^{-1}(1)} \cdots a_{n,\sigma^{-1}(n)},$$
since $\varepsilon(\sigma) = \varepsilon(\sigma^{-1})$.
In this sum, each term corresponds to a permutation σ.
However, as σ ranges over all permutations, so does σ^{-1} because a permutation determines its inverse uniquely.
Hence our sum is equal to
(**) $\sum_\sigma \varepsilon(\sigma)a_{1,\sigma(1)} \cdots a_{n,\sigma(n)}$.
The sum (**) is precisely the sum giving the extended form of the determinant of the transpose of A.

Fig.6 <u>Theorem on the determinant of transpose and its proof</u>

```
theory determinant

  det(A:square)
    := sum P:perm⟨col_size(A)⟩.
         sgn(P) * prod I:seg⟨col_size(A)⟩. A[P[I],I]

  theorem  determinant_of_transpose
     all A:square.  det(A) = det(trans(A))

     proof
       let a:square be arbitrary
       n := col_size(a)
       then n = col_size(trans(a))
       det(a)
       = sum P:perm⟨n⟩. sgn(P) * prod I:seg⟨n⟩. a[P[I],I]
                                      by definition
       = sum P:perm⟨n⟩. sgn(inv(P)) * prod I:seg⟨n⟩. a[inv(P)[I],I]
       = sum P:perm⟨n⟩. sgn(P) * prod I:seg⟨n⟩. trans(a)[P[I],I]
           since
             let p:perm⟨n⟩ be arbitrary
             prod I:seg⟨n⟩. a[inv(p)[I],I]
             = prod I:seg⟨n⟩. a[inv(p)[p(I)],p[I]]
             = prod I:seg⟨n⟩. trans(a)[p[I],I]
                 since
                   let i:seg⟨n⟩ be arbitrary
                   a[inv(p)[p[i]],p[i]]
                   = a[i,p[i]]
                   = trans(a)[p[i],i]
                 end_since
             sgn(inv(p)) = sgn(p)
           end_since
        = det(trans(a))     by definition
     end_proof
   end_theorem

 end_theory
```

Fig.7 Its Corresponding description in PDL

5. PROOF CHECKING SUBSYSTEM

The proof checking subsystem is the kernel of the CAP-LA system. This subsystem consists of five modules (Fig.8), conversion, inference, term rewriting, formula manipulation and rule generation.

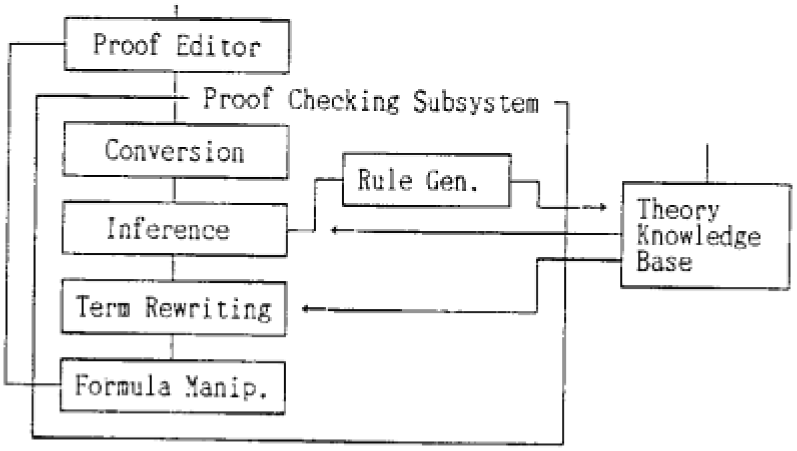


Fig.8 Configuration of the Proof Checking Subsystem

The conversion module is an interface to the proof editor and converts a parsed proof tree into the proof tree, for use in proof checking. The inference module checks the proof tree to see whether each step obeys NK and other inference rules or not. The term rewriting module checks formulas for equality, that is, whether the both sides of the equality in the formula can be rewritten into the same term. The formula manipulation module is used to convert an algebraic formula without changing its meaning. It also has certain knowledge on finite summation (Σ), finite product (Π), and so on. The rule generation module converts the theorem and its proof to rules ready for use in both the proof checking module and the term rewriting module, and inserts them into the theory knowledge base.

In the proof checking module, backward reasoning is performed at in each inference step. For example, given an inference step

$$\frac{\begin{matrix}\vdots & \vdots & \vdots \\ D & E & F\end{matrix}}{G}$$

this module first checks whether or not there exists a strategy to prove G when D, E and F are proved. If not, it checks whether G is verified or not by applying rules in the environment where D, E and F are proved. After checking G, it inserts G in the knowledge base and creates the new environment for checking at next inference step. When a formula with equality must be checked, control is transferred to the term rewriting module. After retrieving possible rewriting rules from the knowledge base, the term rewriting module tries to apply the rules to all redexes in the terms and compares two irreducible terms. If two terms are found equal by this reduction, it returns the result with the environment where they are equal.

6. SYSTEM SESSION —— an example

A sample session with the CAP-LA system is explained in this section.

After entering the CAP-LA system, system menu window appears for function selection. If you want to see a list of theories registered under your name, click the function in the menu, then the next window appears (Fig.9).

This list is also used for selection of the theory you need. If you select a theory "example", its content appears (Fig.10).

If you want to check a text after editing. the text is first checked against the syntax of PDL. If any errors are found, then next window appears (Fig.11).

If all errors are corrected, correctness of the proof is to be checked. During proof checking, you can see at another window the part of theory currently in check. Upon completion of proof checking, if you want to print out the result in a pretty form, you can print it in an English form (Fig.12) or a Japanese form (Fig.13).

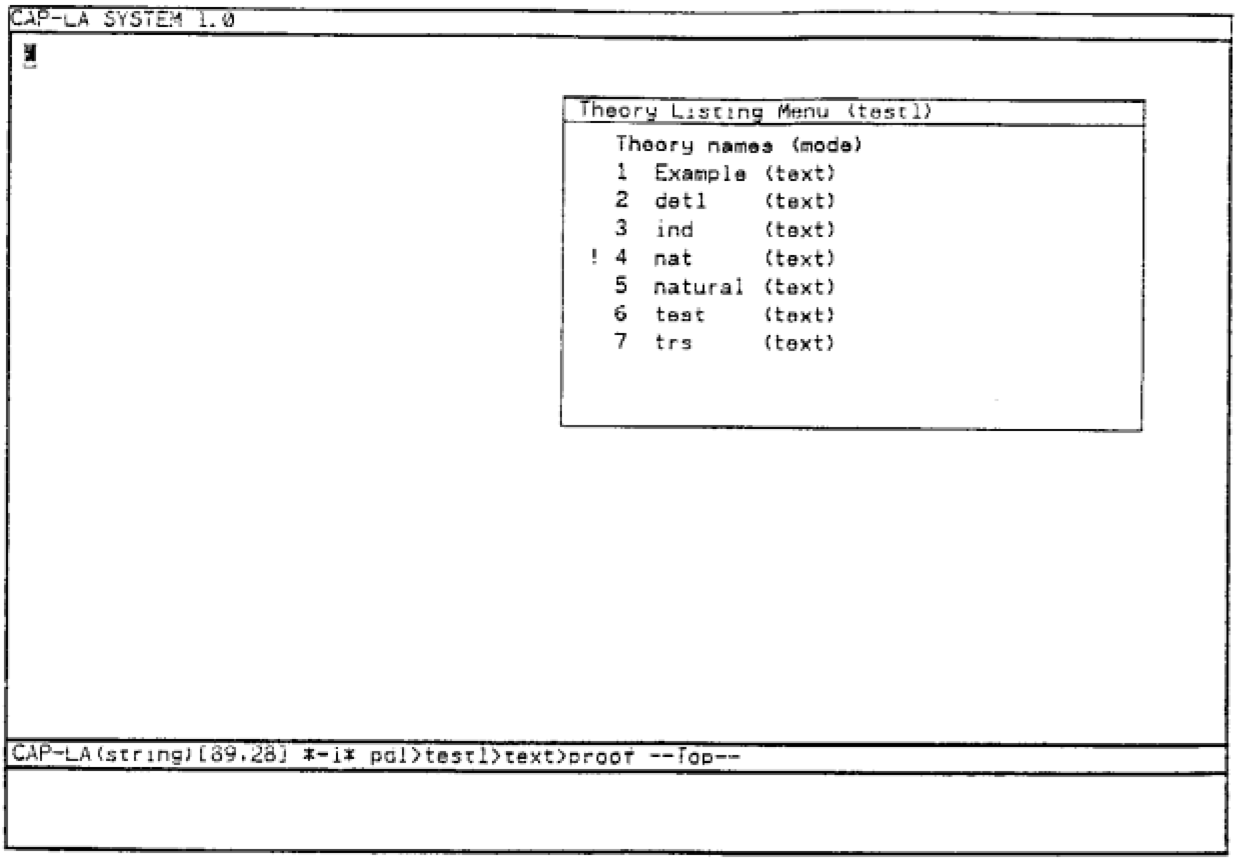
The session proceeds using various windows on a bit-map display (Fig.14).

7. Further Plans for CAP Projects

We recognized from the experience of the first version that PDL can be used to express not only linear algebra but many branches of mathematics and that the performance of proof checking is efficient. So we already began to design the second version of CAP-LA featuring the followings:

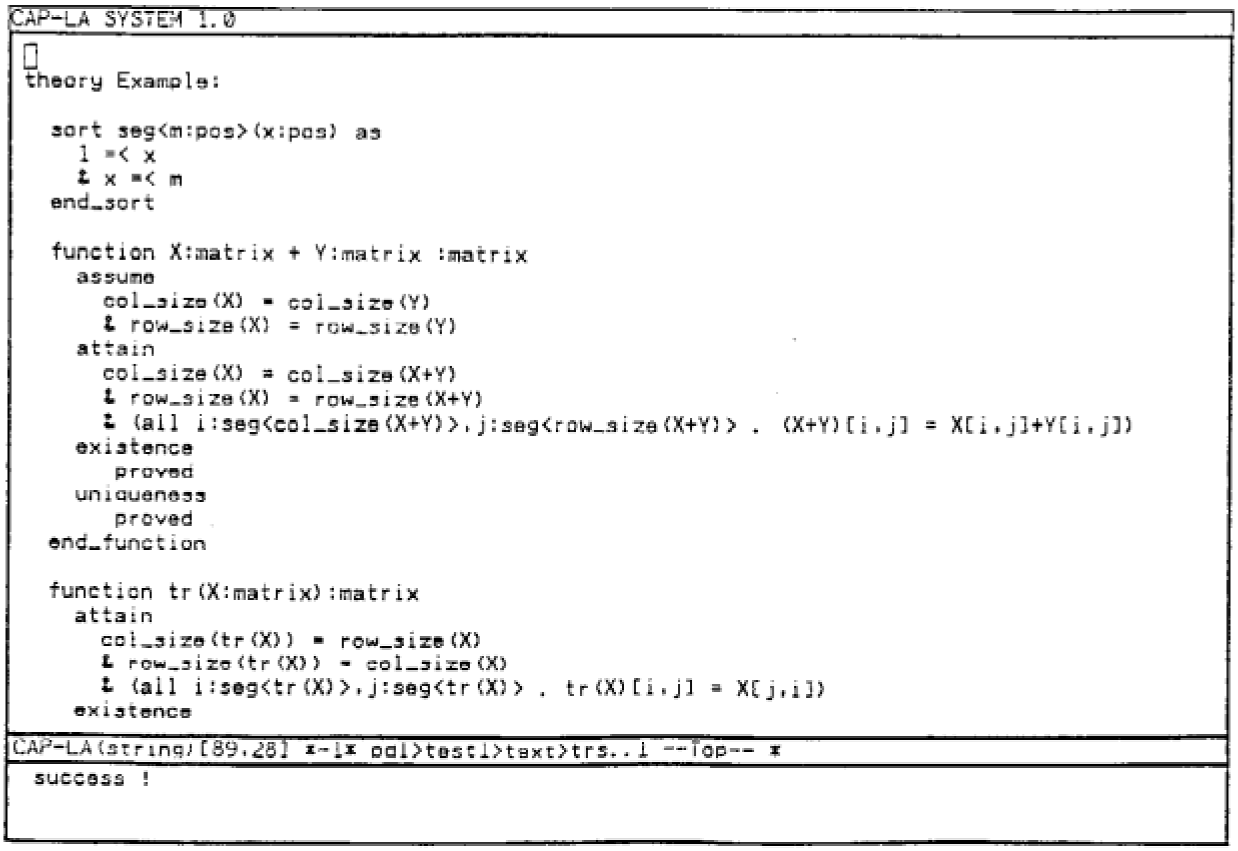
A. PDL

(1) Extension to higher order logic.
(2) Introducing generic types or type variables, with which users can define more natural type hierarchy.
(3) Introducing user-defined proof templates or extraction of proof templates from checked proofs.
(4) Suppression of long, repetitious and tedious proof by admitting 'similarly', '…' and so on in a proof.
(5) Introducing a lot of syntax sugar so that a user can write theorems and proofs easily.



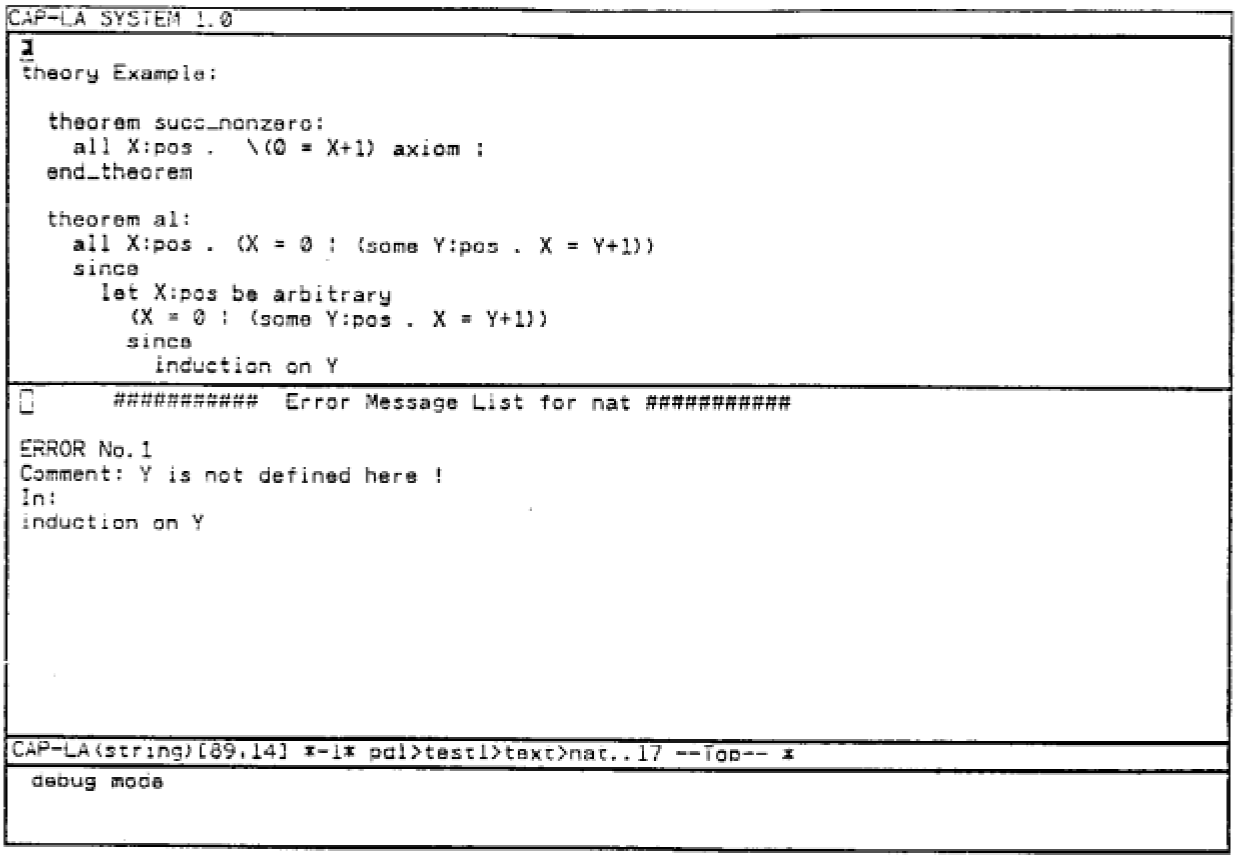
```
CAP-LA SYSTEM 1.0

                                         Theory Listing Menu (test1)
                                            Theory names (mode)
                                            1  Example (text)
                                            2  det1    (text)
                                            3  ind     (text)
                                          ! 4  nat     (text)
                                            5  natural (text)
                                            6  test    (text)
                                            7  trs     (text)

CAP-LA(string)[89,28] *-1* pd1>test1>text>proof --Top--
```

Fig.9



```
CAP-LA SYSTEM 1.0

theory Example:

  sort seg<m:pos>(x:pos) as
    1 =< x
    & x =< m
  end_sort

  function X:matrix + Y:matrix :matrix
    assume
      col_size(X) = col_size(Y)
      & row_size(X) = row_size(Y)
    attain
      col_size(X) = col_size(X+Y)
      & row_size(X) = row_size(X+Y)
      & (all i:seg<col_size(X+Y)>,j:seg<row_size(X+Y)> .  (X+Y)[i,j] = X[i,j]+Y[i,j])
    existence
       proved
    uniqueness
       proved
  end_function

  function tr(X:matrix):matrix
    attain
      col_size(tr(X)) = row_size(X)
      & row_size(tr(X)) = col_size(X)
      & (all i:seg<tr(X)>,j:seg<tr(X)> . tr(X)[i,j] = X[j,i])
    existence
CAP-LA(string)[89,28] *-1* pd1>test1>text>trs..1 --Top-- *
  success !
```

Fig.10

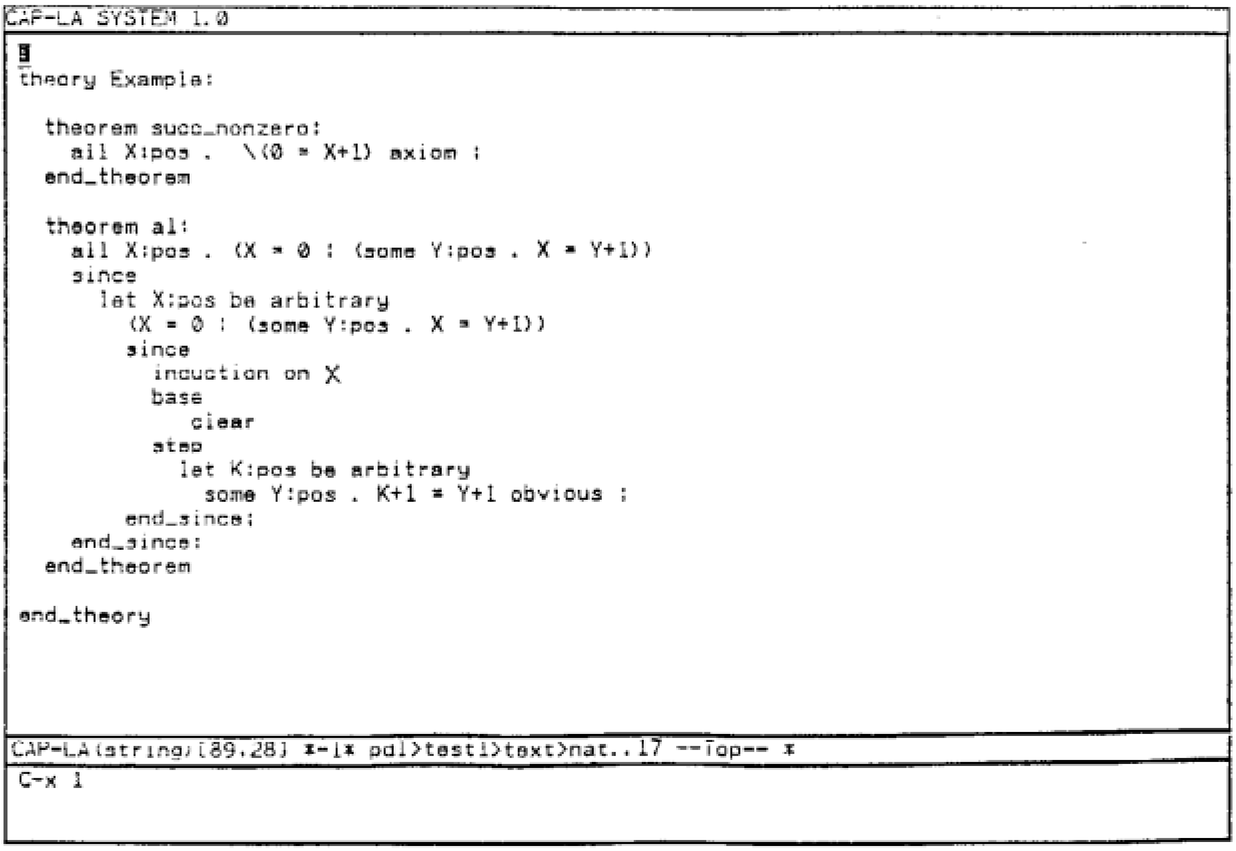


```
CAP-LA SYSTEM 1.0

theory Example:

  theorem succ_nonzero:
    all X:pos .  \(0 = X+1) axiom ;
  end_theorem

  theorem a1:
    all X:pos . (X = 0 ¦ (some Y:pos . X = Y+1))
    since
      let X:pos be arbitrary
        (X = 0 ¦ (some Y:pos . X = Y+1))
        since
          induction on Y
```

```
        ###########  Error Message List for nat ###########

ERROR No.1
Comment: Y is not defined here !
In:
induction on Y
```

```
CAP-LA(string)[89,14] *-1* pdl>test1>text>nat..17 --Top-- *
 debug mode
```

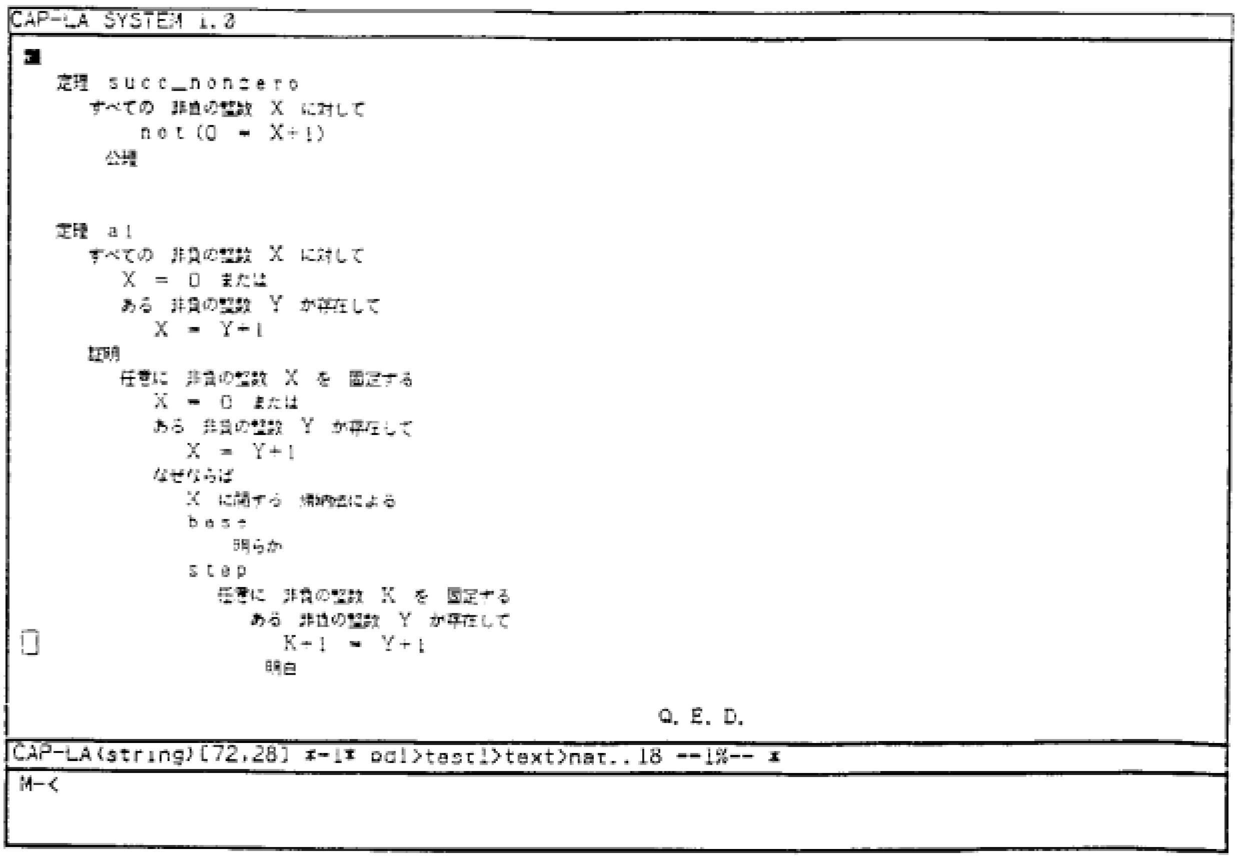
Fig.11



```
CAP-LA SYSTEM 1.0

theory Example:

  theorem succ_nonzero:
    all X:pos .  \(0 = X+1) axiom ;
  end_theorem

  theorem a1:
    all X:pos . (X = 0 ¦ (some Y:pos . X = Y+1))
    since
      let X:pos be arbitrary
        (X = 0 ¦ (some Y:pos . X = Y+1))
        since
          induction on X
          base
             clear
          step
            let K:pos be arbitrary
              some Y:pos . K+1 = Y+1 obvious ;
        end_since;
    end_since:
  end_theorem

end_theory
```

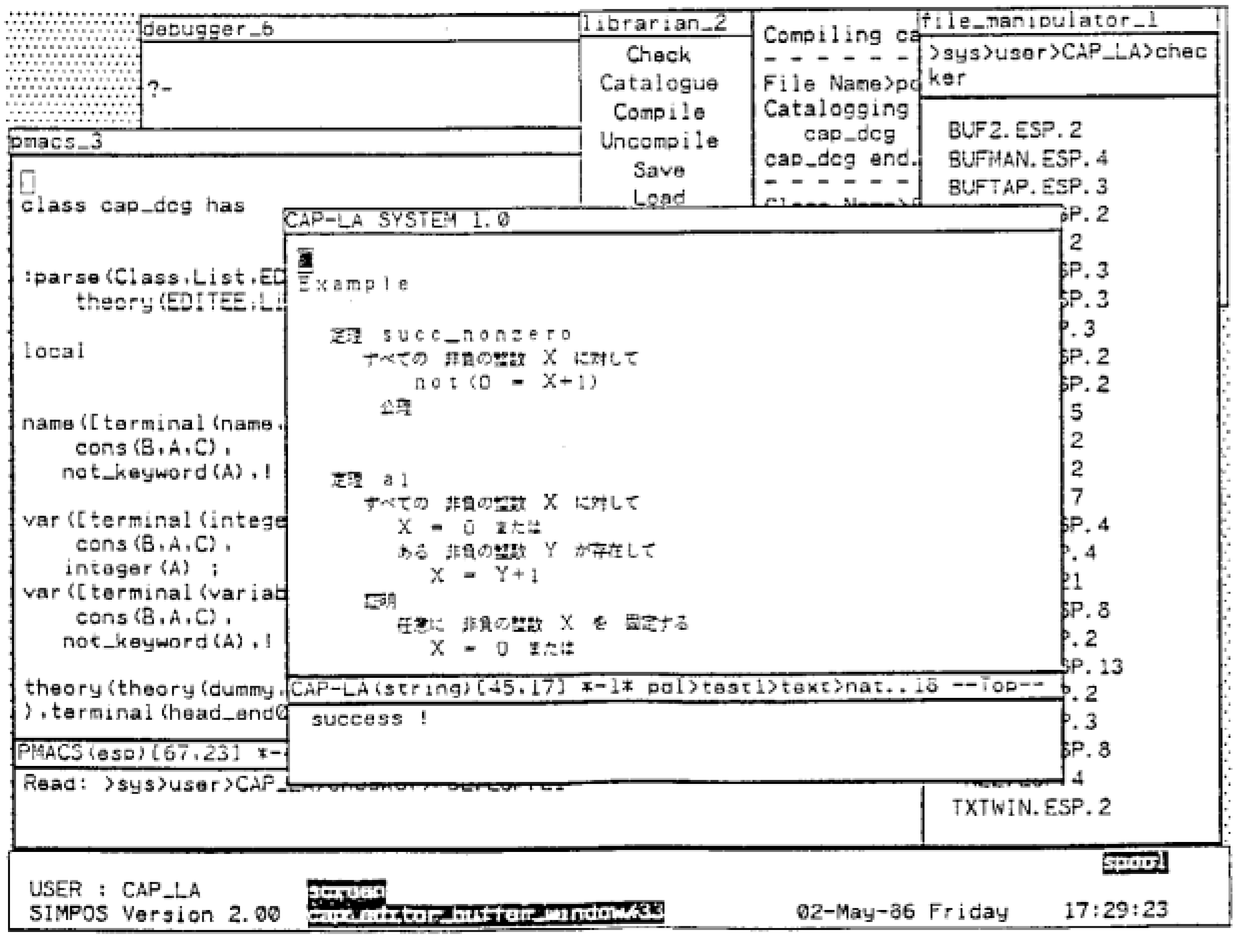
```
CAP-LA(string)[89,28] *-1* pdl>test1>text>nat..17 --Top-- *
C-x 1
```

Fig.12



```
CAP-LA SYSTEM 1.0

  定理 succ_nonzero
    すべての 非負の整数 X に対して
       not(0 = X+1)
     公理


  定理 a1
    すべての 非負の整数 X に対して
       X = 0 または
       ある 非負の整数 Y が存在して
          X = Y+1
    証明
       任意に 非負の整数 X を 固定する
          X = 0 または
          ある 非負の整数 Y が存在して
             X = Y+1
          なぜならば
             X に関する 帰納法による
             base
                明らか
             step
                任意に 非負の整数 K を 固定する
                   ある 非負の整数 Y が存在して
                      K+1 = Y+1
                    明白

                                                 Q. E. D.
CAP-LA(string)[72,28] *-1* pdl>test1>text>nat..18 --1%-- *
M-<
```

Fig.13



```
debugger_6
?-

pmacs_3
class cap_dcg has

:parse(Class,List,ED
    theory(EDITEE,Li

local

name([terminal(name,
    cons(B,A,C),
   not_keyword(A),!

var([terminal(intege
    cons(B,A,C),
   integer(A) ;
var([terminal(variab
    cons(B,A,C),
   not_keyword(A),!

theory(theory(dummy,
),terminal(head_end0
PMACS(esp)[67,23] *-
Read: >sys>user>CAP_
```

```
librarian_2
  Check
  Catalogue
  Compile
  Uncompile
  Save
  Load
```

```
Compiling ca
- - - - - -
File Name>po
Cataloging
   cap_dcg
cap_dcg end.
- - - - - -
```

```
file_manipulator_1
>sys>user>CAP_LA>chec
ker

  BUF2.ESP.2
  BUFMAN.ESP.4
  BUFTAP.ESP.3
  SP.2
  2
  SP.3
  SP.3
  P.3
  SP.2
  SP.2
  5
  2
  2
  7
  SP.4
  P.4
  21
  SP.8
  P.2
  SP.13
  P.2
  P.3
  SP.8
  4
  TXTWIN.ESP.2
```

```
CAP-LA SYSTEM 1.0

Example

  定理 succ_nonzero
    すべての 非負の整数 X に対して
       not(0 = X+1)
     公理


  定理 a1
    すべての 非負の整数 X に対して
       X = 0 または
       ある 非負の整数 Y が存在して
          X = Y+1
    証明
       任意に 非負の整数 X を 固定する
          X = 0 または
CAP-LA(string)[45,17] *-1* pdl>test1>text>nat..18 --Top--
 success !
```

```
USER : CAP_LA                  [illegible]                                       spool
SIMPOS Version 2.00            cap_editor_buffer_window433     02-May-86 Friday     17:29:23
```

Fig.14

B. Proof Checking

(1) Combination of forward and backward reasoning.
(2) Type inference and type checking for proof checking.
(3) Utilizing conjunctions, adjectives and adverbs in a proof for the improved efficiency of proof checking.
(4) Checking the higher order theorems or metamathematical theorems.

C. User Interface

(1) Two-dimensional display to display such as Σ, Π, matrices, etc., using a bit-map display.
(2) Fully interactive proof checking.
(3) Stepwise refinement of a proof by collaboration with the system.
(4) Detailed error messages and help messages for the novice user and more natural English and Japanese proof forms.

Other CAP subprojects, CAP-SDG and CAP-QJ, are also active in connection with CAP-LA. At first we selected three branches of mathematics, but now, judging from the experience with CAP-LA, we do not think it necessary to restrict the scope (of the system) to some specific branches of mathematics.

CAP-SDG, which does not necessarily mean synthetic differential geometry, aims at a fully interactive proof checker based on the term rewriting system. The term rewriting system considers both terms and formulas as terms, and handles higher order logic easily. We plan to develop the second version of CAP-LA and the prototype system of CAP-SDG this year, and integrate them next year.

We are currently under experimentation with CAP-QJ to describe some metamathematical theorems and proofs in the QJ system. A compiler and an interpreter for the programming language Quty are scheduled to be implemented this year. This subproject is closely related to another IPS project, called Program Construction System (PCS), which aims to derive programs from proofs. Given a constructive proof for a formal specification written in QJ, a realizability interpreter derives a realizer as an executable part of the proof, that is, a program. The realizability interpreter is also now under experimentation and this system will employ Martin-Lof's intuitionistic type theory as generalization and specialization of program modules.

8. CONCLUSIONS

The FGCS project in Japan did not select Lisp as a kernel language, but a logic programming language like Prolog because of its powerful functions based on unification and its high productivity. The logic programming language itself is based on theorem proving such as SLD resolution. Studies of theorem proving or proof checking play an essential role in the total plans of FGCS in Japan. Computer science has a history of only 40 years, while history of mathematics lasted over 2,000 years. I expect our approach to mechanization of mathematics will bring forth many contributions to FGCS project, for instance, the accumulated knowledge on knowledge, programming and inference.

While there are many systems for automated theorem proving and proof checking [10,11,12,13,14], most of them are stand alone systems and seem to be limited in their aims. In Japan, there were not many studies in the field when we decided to start our project. Now, CAP project started in the FGCS framework, and is planned to become a kernel system of many knowledge processing systems. The CAP-LA system is intended to be the first step to such a system.

Acknowledgement

We would like to thank members of Second Laboratory of ICOT and its CAP Working Group for useful discussion and suggestion. And we would also like to express our gratitude to engineers who have participated in the development of CAP-LA.

References

[1] T.Ida, M.Sato, S.Hayashi, M.Hagiya et al, "Higher Order: its Implication to Programming Languages and Computational Models", ICOT TM-29, 1983.
[2] ICOT Working Group 5, "Several Aspects on Unification", ICOT TM-46, 1984.
[3] M.Sato, "Typed Logical Calculus", TR-85-13, Dept. of Computer Science,Fac. of Science, Univ. of Tokyo, 1985.
[4] M.Sato and T.Sakurai, "Qute: A Functional Language based on Unification", Proceedings of the International Conference on Fifth Generation Computer Systems, pp157-165, 1984.
[5] A.Kock, Synthetic Differential Geometry, London Mathematical Society Lecture Note Series 51, Cambridge University Press, pp311, 1981.
[6] S.Hayashi, "Towards Automated Synthetic Differential Geometry 1 --- basic categorical construction", ICOT TR-104, 1985.
[7] S.Uchida and T.Yokoi, "Sequential Inference Machine: SIM Progress Report", ICOT TR-86, 1984.
[8] T.Yokoi and S.Uchida, "Sequential Inference Machine: SIM Its Programming and Operating System", ICOT TR-87, 1984.
[9] T.Chikayama, "Unique Feature of ESP", Proceedings of the International Conference of the Fifth Generation Computer System, Tokyo, pp292-298, 1984.
[10] M.J.Gorden, A.J.Milner and C.P.Wadsworth, "Edinburgh LCF", Lecture Notes in Computer Science,78, Springer, 1979.
[11] J.Ketonen and J.S.Weening, "EKL — An Interactive Proof Checker, User's Reference Manual", Department of Computer Science, Stanford Univ., 1983.
[12] M.Hagiya and S.Hayashi, "Some Experiments on EKL", ICOT TM-101, 1985.
[13] A.Trybulec and H. Blair, "Computer Assisted Reasoning with Mizar", IJCAI'85, pp26-28, 1985.
[14] N.Shanker, "Towards Mechanical Metamathematics", Journal of Automated Reasoning, 1, pp407-434, 1985.