

TR-426

Potentials of General-Purpose Reasoning
Assistant System EUODHILOS

by

H. Sawamura, T. Minami,
K. Sato and K. Tsuchiya
(Fujitsu)

October, 1988

©1988, ICOT

ICOT

5th Fl. Kokusai Bldg. 21F
1-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

Potentials of General-Purpose Reasoning Assistant System EUODHILOS

*Every universe of discourse has its logical structure.
S. K. Langer (1925)*

HAJIME SAWAMURA, TOSHIRO MINAMI

*International Institute for Advanced Study of Social Information Science (IIAS-SIS),
FUJITSU LIMITED, 140 Miyamoto, Numazu, Shizuoka 410-03, JAPAN
hajime@iias.fujitsu.junet tos@iias.fujitsu.junet*

KAORU SATO AND KYOKO TSUCHIYA

FUJITSU LABORATORIES LTD., 1015 Kamikodanaka, Nakahara-ku, Kawasaki, Kanagawa 211, JAPAN

(draft)

Summary

Much work has been done on special-purpose reasoning assistant systems whose underlying logics are fixed. In contrast with such a trend, this paper is devoted to a new dimension of computer-assisted reasoning research, that is, a general-purpose reasoning assistant system that allows a user to define his own logical system relevant for the objects in the problem domain and to reason about them.

In the first half of the paper, the need, significance and design principle of EUODHILOS : a general-purpose system for computer-assisted reasoning, are discussed, then the system overview is described, placing emphases on the following three points : (1) formal system description language, (2) proving methodology based on several sheets for logical thought, (3) visual human-computer interface for reasoning. In the latter half, the potentials and usefulness of EUODHILOS are demonstrated through experiments and experiences of its use by a number of logics and proof examples therein, which have been used or devised in computer science, artificial intelligence and so on.

INTRODUCTION

A new dimension of computer-assisted reasoning research is being explored in this paper. It aims at a general-purpose reasoning assistant system that allows a user to interactively define the syntax and inference rules of a formal system and to construct proofs in the defined system.

We have named such a system EUODHILOS, an acronym reflecting our philosophy or observation that *every universe of discourse has its logical structure*, which turns out to spell and sound like a Greek philosopher's name.

In these days, various logics play important and even essential roles in computer science and artificial intelligence (e.g., [Genesereth 87], [Smets 88]), and surprisingly in aesthetics which is thought of as being in a directly opposite position to logic (e.g., [Langer 25], [Kunst 76], [Rahn 79]). Specifically, it can be said that they provide expressive devices for objects and their properties, and inference capabilities for reasoning about them. It is also the case that symbols manipulating methods provided in logics are basically common to all scientific activities. So far, they have made use of a wide variety of logics, including first-order, higher-order, equational, temporal, modal, intuitionistic, relevant, type theoretic logics and so on. However, implementing an interactive system for developing proofs is a daunting and laborious task for any style of presentation of these logics. For example, one must implement a parser, term and formula manipulation operations (such as substitution, replacement, juxtaposition, etc.), definitions, inference rules, rewriting rules, proofs, proof strategies and so on, depending on the logic to be needed. Thus, it is desirable to find a general theory of logics and a general-purpose reasoning assistant system that captures the uniformities of a large class of logics so that much of this effort can be expended once and for all. We aim at building an easy to use and general reasoning system which handles as many of these logics as possible.

There are three major subjects to be pursued for such an interactive and general reasoning supporting system. One is, of course, a language expressive enough to describe a large class of logics. The second is the kind of reasoning styles suitable for human reasoners which should be taken into account. More generally, reasoning (proving) methodology, which reminds us of programming methodology, needs to be investigated. The third subject is reasoning-oriented human-computer interface that may be well established as an aspect of reasoning supporting facilities. An easy to use system with good interface would be helpful for one to conceive ideas in reasoning.

We believe that a general-purpose reasoning assistant system incorporating these points should cater to the mathematician or programmer who wants to do proofs, and also to the logician or computer theorist who wants to experiment with different logical systems.

The remainder of this paper is organized as follows. In the first half of the paper, following the discussion of the need, significance and design philosophy of EUODHILOS, a system summary of EUODHILOS under development is described, where we place emphases on the following three points: (1) formal system description language, (2) proving methodology based on several sheets for logical thought, (3) visual human-computer interface for reasoning. In the latter half, the potentials and usefulness of EUODHILOS are shown through experiments and experiences of its use by a number of logics and proof examples therein, which have been used or devised in computer science, artificial intelligence and so on. They include a logical puzzle and the halting problem with the first-order logic, propositional modal logic, intuitionistic

type theory, program verification with Hoare logic, and a reflective proof with intensional logic.

NEED, SIGNIFICANCE AND DESIGN PHILOSOPHY

Much work has been devoted to special-purpose reasoning assistant systems whose underlying logics are fixed. At this stage, why are we about to pursue or explore a new dimension such as a general-purpose reasoning assistant system ?

We first take up some issues concerned with the generality in reasoning assistant system and several aspects to view such a generality. We have already seen that it can be easily found and recognized that in these days a logic or logical methodology is forming a kind of paradigm for promoting computer science, artificial intelligence and so on. (e.g., [Genesereth 87], [Smets 88], [Langer 25], [Kunst 76], [Rahn 79]). And we stated that it is desirable to find a general theory of logics and a general-purpose reasoning assistant system that captures the uniformities of a large class of logics so that much effort for providing reasoning facilities can be expended once and for all, and hence we aim at building an easy to use and general reasoning system which handles as many of these logics as possible. This was our first motivation for pursuing the generality in reasoning assistant system. The second issue comes from the rigorous approach to program construction. Abrial [Abrial 84] claims that a general-purpose proof checker be perhaps one of a set of tools for computer aided programming when we consider program construction from various theories. We are certainly in a situation that before embarking on the construction of a program we need to study its underlying theory, that is to give a number of definitions, axioms and theorems which are relevant to the problem at hand. The third issue is concerned with construction of a logical model, or more generally methodology of science. We observe that the whole phases of human reasoning process consist of the following three phases : (1) making mental images about the objects or concepts, (2) making logical models which describe the mental images, (3) examining the models to make sure that they coincide with mental images. It is not conceivable that the phase (1) could be aided mechanically since some part of the phase (1) is very creative. On the other hand, it is very likely that the phases (2) and (3) are largely supported mechanically by allowing to modify or revise the definition of the language used for the modeling and by introducing certain reasoning devices. These are just the points that a general-purpose reasoning assistant system is intended to support.

A philosophical aspect of the generality from a logical point of view can be found in [Langer 25] and Wittgenstein's philosophy. Langer stated that "every universe of discourse has its logical structure". It eventually supports our discussions about the need and significance of the generality in reasoning assistant system.

Taking these needs, observations and philosophy into consideration, fundamental design principles of EUODHILOS are set up as follows :

- Realization of a general reasoning system, reflecting the philosophy that every universe of discourse has its logical structure.

- Support of logical thought, symbolic or logical manipulations done by human reasoners

- Provision of an easy to use environment for supporting proof constructions

- Environment for experimenting logical model construction and methodology of science

These requirements lead to the research and development of general-purpose reasoning assistant system EUODHILOS with the following outstanding features :

- Formal system description language based on DCG

- Proving methodology using sheets of thought

- Reasoning-oriented human-computer interface

In what follows, we briefly sketch a current version of EUODHILOS.

OVERVIEW OF EUODHILOS

Functional features

We list the main features of EUODHILOS and explain them briefly (see [Sato 88a] and [Sato 88b] for the details). We start by describing the language of a logic to EUODHILOS. Fundamentally, EUODHILOS has almost no defaults, so it must be told everything,

Formal system description language

What on earth is a logic ? Or what language should be expressive enough to describe or deal with logics ? The answers to these questions, in general, could turn out to settle the formal system description language for capturing the uniformities of a large class of logics so that it can be used as the basis for implementing proof systems. There have been some attempts to pursue it, sharing a common goal with our EUODHILOS (e.g., Prolog is employed as a logic description language in [Sawamura 86], λ Prolog in [Felty 88] and [Miller 87], Martin-Löf's intuitionistic type theory in [Harper 87] and [Griffin 87], a specification language for a wide variety of logics in [Abrial 84], attribute grammar in [Reps 84] and ML in [Gordon 82]).

Contemporary logics, in general, may be considered as having a logical framework consisting of proof theory and model theory. Proof theory is to specify a syntactical part of logic and model theory a semantical part of logic. In this paper we are mainly concerned with specifying the syntactical aspect of logic. From a syntactic point of view, a formal system (logical system), in general, is supposed to be specified by the two constituents : Language system and derivation system.

(1) Language system

A language is a tool for talking about objects and is constructed from underlying primitive symbols. It is usually specified by utilizing some of the following items: variable, constants and function as individual symbols, predicate (including equality), logical connective, auxiliary symbol. The attributes such as type, sort, arity, operator precedence are sometimes associated with some of these symbols. Once these primitive symbols are specified, complexies such as terms, formulas, etc., are constructed from them by fomation rule. Besides, mechanisms for defining or abbreviating symbols, are usually required. One of the main questions that may be raised at this point is the following : what metalanguage is natural and sufficient enough to describe such an object language ?

(2) Derivation system (or deductive theory)

Derivation system gives us a means to manipulate logical languages, which are specified by axioms, inference rules, derived rules, rewriting rules, and concepts of proofs, etc. Insofar as we are confined ourself to the existing types of formal system, we can enumerate primitive operations included in them, for instance, substitution, replacement, juxtaposition, detachment, renaming, unification, instantiation, etc. are common operations among various logics except for the difference of languages. Since we consider a general-purpose reasoning system for logics, we have to provide a general method for such symbol manipulations. So, one of the main questions that may be raised at this point is the following : what sort of primitive operations should be sufficient to manipulate logics and how they can be provided in a generic manner ?

In addition to them, we have to pay attention to the following concepts proper in logics, such as "free" and "bound", "something is free for a variable in an expression", etc., although these can be often dealt with in a recursive fashion.

Expressiveness of definite clause grammar and implication to defining logics

In EUODHILOS, a language system to be used is designed and defined by a user, a current solution for formal system description language is to employ so called definite clause grammar (DCG) [Pereira 80], where the problem of recognizing, or parsing a string of a language is then transformed into the problem of proving that a certain theorem follows from the definite clause axioms which describe the language. The DCG formalism for grammars is a natural extension of context-free grammar (CFG). As such, DCG inherits the properties which makes CFG so important for language theory such as the modularity of a grammar description and the recursive embedding of phrases which is characteristic of almost all interesting languages, including languages of logics. It is, however, well known that CFG is not fully adequate for describing natural language, nor even many artificial languages. DCG overcomes this inadequacy by extending CFG in the following three points [Pereira 80].

- (i) context-dependency
- (ii) parameterized nonterminal
- (iii) procedure attachment

These also yield great advantages for specifying logical grammars, compared with those mentioned above. DCG provides for context-dependency in a grammar, so that the permissible forms for a phrase may depend on the context in which that phrase occurs in the string. DCG is somewhat similar to attribute grammar in the sense that context free grammar is made context sensitive by associating with grammar rules a semantical facility [Reps 84]. The necessity of context-dependency is often encountered in defining logical syntax. The following examples show how naturally and economically DCG allows us to express context-dependency occurring in the ordinary logical practice and allows arbitrary tree structure to be build in the course of the parsing, with the help of (ii) and (iii).

The defining clause of first-order terms such as "If f is a function symbol of arity 2 and t and s are terms, then $f(t, s)$ is a term" is represented as

$\text{term}(f(T,S)) \rightarrow \text{functor}(f), "(", \text{term}(T), ",", \text{term}(S), ")", \{\text{arity}(f) = 2\}.$

The defining clause of terms in intensional logic such as "If A is a term of type (a, b) and B a term of type a , then $A \bullet B$ is a term of type b " is represented as

$\text{term}(A \bullet B, b) \rightarrow \text{term}(A, a), \text{term}(B, b).$

Once a definite clause definition for a logical syntax have been given, then the bottom-up parser [Matsumoto 83] and unparser for the defined language are automatically generated, which are to be internally used in all the phases of symbol manipulations. The reason why we do not generate a top-down parser for the defined language is as usual to avoid the anomaly of left-recursiveness which often appears in the ordinary definition of a logical syntax. The internal structures of the expressions of the language are automatically constructed just at the same time as the automatic generation of the parser and unparser. These functions greatly lighten a user's burden in setting up his own language.

A derivation system consists of an inference system and a rewriting system. They are given in a natural deduction style presentation by a user. Especially, an inference rule is stated as a triple consisting of three elements, where the first is the derivations of the premises of a rule, the second the conclusion of a rule, and finally the third the restrictions that are imposed on the derivations of the premises, such as variable occurrence condition (eigenvariable) and substitutability such as " a is free for x ". Well-known typical styles of logic presentations such as Hilbert's style, Gentzen's style, Equational style could be treated within this framework.

Inference rules are presented in terms of the schematic rule description language in a natural deduction style as follows :

[Assumption ₁]	[Assumption ₂]	...	[Assumption _n]
⋮	⋮		⋮
Premise ₁	Premise ₂		Premise _n
<hr/>			
Conclusion			

where brackets are used to encompass an assumption to be discharged. ":" denotes a sequence or a subtree of formulas which is a part of a proof from the assumption and each assumption is optional. If a premise has the assumption, its subtree of a proof indicates a conditional derivation. An inference rule may be permitted to apply if all the premises are obtained in this manner and the application condition is satisfied. Similarly, rewriting rules are presented in the following schematic format :

$$\frac{\text{exp}_1}{\text{exp}_2}$$

where exp_1 and exp_2 denote the subexpressions occurring in the expressions before and after the rewriting of an expression respectively.

Proof constructing facilities

The major drawback of reasoning in formal logic is that derivations tend to be lengthy and tedious because of the detailed level of derivations to be required in reasoning. Furthermore, performing formal derivations is time-consuming and error-prone. The readers may notice that such a situation is quite similar to the one in the formal development of programs in which programs can be derived or transformed and properties of programs can be established.

Using computers for formal reasoning is expected to overcome the problems with errors and the time-consuming task. The current version of EUODHILOS has the following unique facilities which support constructions of proofs in the defined formal system.

(1) Sheets of thought (or proof sheet)

This originated from a metaphor of work or calculation sheet and is apparently analogous to the concept of sheet of assertion which is due to C. S. Peirce [Peirce 74]. A sheet of thought, in our case, is supposed to be a field of thought where we are allowed to draft a proof, to compose proof fragments or detach a proof, or to reason using lemmas, etc., while a sheet of assertion is a field of thought where existential graph as an icon of thought is supposed to be drawn. Obviously, proving by the use of a sheet of thought yields proof modularization considered to be useful for proving in large, which is analogous to the concept of program modularization, to borrow the term of software engineering.

Technically, a sheet of thought is a window with multi-functions for reasoning in the multi-window environment of a Personal Sequential Inferential machine (PSI).

(2) Tree-form proof

As mentioned above, inference and rewriting rules are presented in a natural deduction style. This naturally induces a construction of a proof into a tree-form proof. Consequently it leads to representing a proof structure explicitly, in other words, proof visualization.

(3) Proving methodology

It is desirable that reasoning or proof construction can be done along the natural way of thinking of human reasoners. Therefore EUODHILOS supports the typical method for reasoning, that is, top-down reasoning, bottom-up reasoning and reasoning in a mixture of them. They are accomplished interactively on several sheets of thought.

It is planned to incorporate not only such a proving methodology but also methodology of science (e.g., Lakatos' mathematical philosophy of science [Lakatos 76], Kitagawa's relativistic logic of mutual specification [Kitagawa 63], etc.).

As an example of deduction process on sheets of thought, let us illustrate how one can proceed a deduction by using connection and separation functions of sheets of thought. In order to deduce forward by applying an inference rule, one has to start by selecting the formulas used as premises of the rule. Then one may select an appropriate inference rule from the rule menu which has been automatically generated at the time of logic definition, or he may input a formula as the conclusion. If one selects a rule, then the system applies to the rule to the premises and derive the conclusion. If he gives the conclusion, then the system searches the rules and tries to find one which coincides with this deduction. In the case of backward reasoning, the reasoning process is converse to the forward reasoning, so that the intermediate proof may turn out to be separated into partially justified proof fragments and the complete justification of those partially justified proof fragments is delayed to the completion of a final proof tree.

(i) Connection

(a) Connection by complete matching : Two proof fragments can be connected through a common formula occurring in them when one of them is a hypothesis and the other a conclusion. The process begins by selecting the two formulas and invoking the proper operations. As a result, the proof fragments are connected into the one proof fragment. Schematically, This amounts to attaining the following inference figure which can be viewed as valid :

$$\begin{array}{ll} \Gamma \vdash C_i & \text{(on a sheet of thought)} \\ \Delta, C_i, \Sigma \vdash A & \text{(on a sheet of thought)} \end{array}$$

$$\Gamma, \Delta, \Sigma \vdash A \quad \text{(on a sheet of thought)}$$

where Γ , Δ and Σ represent sequences of formulas (possibly empty), and A and C_i denote formulas in some defined logical system.

(b) Connection by the use of a rule of inference : This is essentially a forward reasoning and may be called a distributed forward reasoning. The process is similar to the above except that the connection is done from the distributed proof fragments through an appropriate rule of inference. Let us take an example schema of modus ponens :

$\Gamma \vdash A \equiv B$	(on a sheet of thought)
$\Delta \vdash A$	(on a sheet of thought)

$\Gamma, \Delta \vdash A$	(on a sheet of thought)
---------------------------	-------------------------

with same proviso, adding that B represents a formula. Besides, connection methods such as analogical matching, instantiation, etc., would become extremely beneficial to intelligent reasoning system, which are left as future subjects.

(ii) Separation

The separation is the converse to the connection by complete matching. The separation process begins by selecting a formula occurring in a sheet of thought and invoking the proper operations. As a result, the proof fragment are detached into the two fragments. Schematically, This amounts to the converse to the connection by complete matching above. So we omit it.

Human-computer interface for reasoning

In order to make the system user-friendly and easy to use, we have paid much attention to the visualization of interface. In EUODHILOS the following facilities are available as human-computer interface for ease in communicating and reasoning with a computer, in particular facilities for inputing formulas and formula visualization.

(1) Formula editor

This is a structure editor for logical formulas and makes it easy to input, modify and display complicated formulas. In addition to ordinary editing functions, it provides some proper functions for formulas such as rewriting functions.

(2) Software keyboard and Font editor

These are used to make and input special symbols often appearing in various formal systems. It is a matter of course that provision of special symbol which reasoners are accustomed to use makes it possible to reason as usual on a computer.

(3) Stationery for reasoning

Independently of a logic under consideration, various reasoning tools such as decision procedures become helpful and useful in reasoning processes. In a sense it may also play a role of a model which makes up for a semantical aspect of reasoning. Currently, a calculator for Boolean logic is realized as a desk accessory.

The screen layout in Fig. 1 shows a proof in which formula editor and software keyboard are being used.

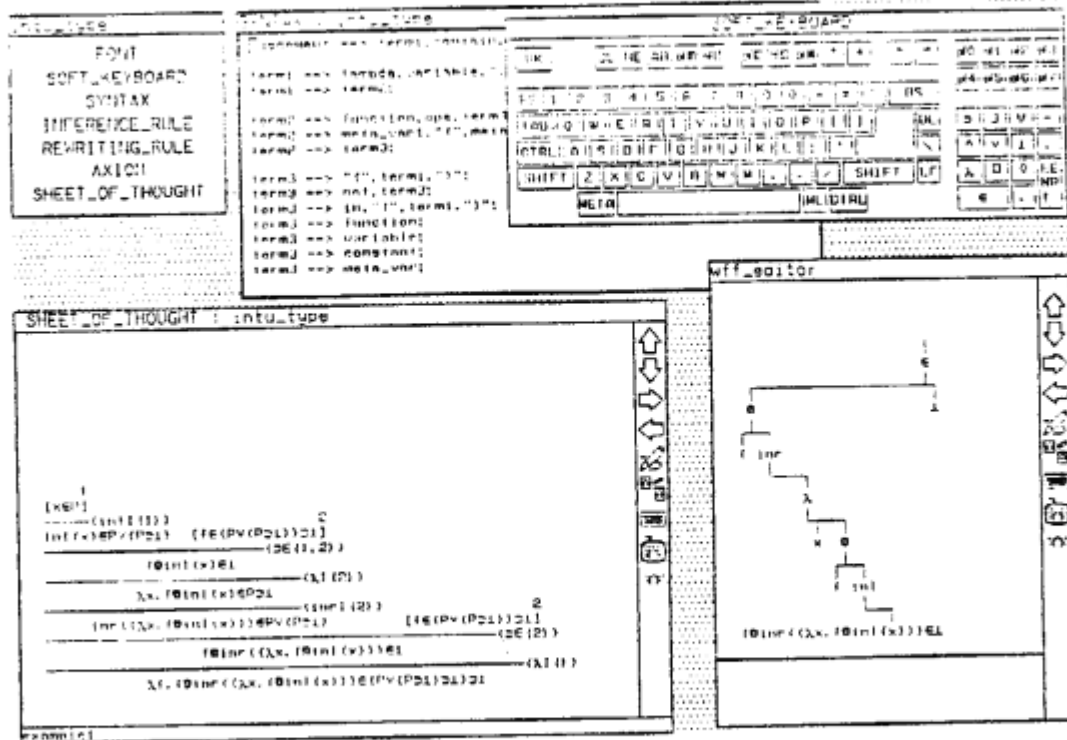


Fig. 1 Reasoning-oriented human-computer interface

Implementation

Exploiting the bit-map display with multi-window environment, mouse, icon, pop-up-menu, etc., EUODHILOS is implemented in ESP language on PSI-II/SIMPOS, and its current size is about 5MB. The system configuration of EUODHILOS is illustrated in Fig. 2. The system consists of two major parts ; one for defining a user's logical system and the other for constructing proofs on sheets of thought.

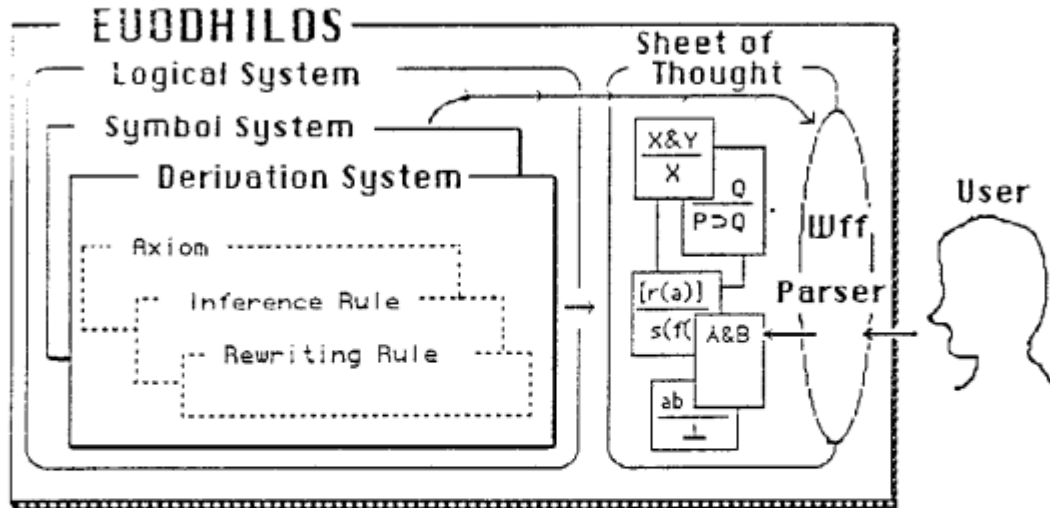


Fig. 2 An illustration of EUODHILOS

EXPERIMENTS AND EXPERIENCES WITH EUODHILOS

We have tried to apply EUODHILOS to various types of reasoning. Logics and proof examples therein that we have dealt with so far on EUODHILOS include various pure logical formulas, unsolvability of the halting problem and an inductive proof with first-order logic (NK), modal reasoning of program with propositional modal logic (T), reflective proof of a metatheorem with intensional Logic (IL), Martin-Löf's intuitionistic type theory, program verification with Hoare logic.

In this section, first taking up two representative formal systems from the circle of program semantics and program verification, we show how EUODHILOS can be used to specify a logic and construct a proof under the specified logic, together with their typical screen layouts. In the last part, we list some other proof experiments with different logics. The important point here is not the complexity of the examples, but rather the holistic understanding of a whole story played with EUODHILOS. These proof experiments with different logical systems could help to convince the readers of the potential and usefulness of EUODHILOS in a much wider range of applications.

Martin-Löf's intuitionistic type theory

The first reasoning system we have chosen as an example is a tiny subset of constructive type theory described in [Martin-Löf] and [Backhouse 88]. We try to prove the following theorem : $\sim\sim(P \vee \sim P) \quad (\equiv (P \vee (P \supset \perp) \supset \perp) \supset \perp)$, which means that the law of excluded middle cannot be refuted, and is an instance of Glivenko's theorem that if P is any tautology of the classical propositional calculus then the proposition $\sim\sim P$ is always constructively valid. We set

up the language for proving the theorem first in terms of BNF for reference and then its DCG description.

Tiny language for type theory (BNF)

```

<judgement> ::= <term> ∈ <type>
<term> ::= <variable> | <constant> | ¬<term> | <function>•<term> | λ<variable>.<term> | inl(<term>) |
      inr(<term>) | <meta-term>
<variable> ::= x
<constant> ::= a | b
<function> ::= f
<type> ::= <basic-type> | <type> ∨ <type> | <type> ⊃ <type> | <meta-type>
<basic-type> ::= P | ⊥
<meta-term> ::= F
<meta-type> ::= A | B

```

Tiny language for type theory (DCG)

The following DCG definition may be somewhat tedious and roundabout for the reasons of the abilities of the current bottom-up parser and unparser generators. However, it will be improved so as to be more natural, by adding operator precedence to DCG.

Syntax of object language :

```

judgement --> term, in, type ;
term --> lambda, variable, ".", term1 ;
term --> term1 ;
term1 --> term1, apply, term2 ;
term1 --> term2 ;
term2 --> "(", term, ")" ;
term2 --> or-intro, "(", term, ")" ;
term2 --> variable ;
term2 --> constant ;
in --> ∈ ;
apply --> "•" ;
lambda --> "λ" ;
or-intro --> "inr" ; "inl" ;
variable --> x ; f ; a ; b ;
type --> type1, imply, type ;
type --> type1 ;
type1 --> type2, or, type1 ;
type1 --> type3 ;
type3 --> "(", type, ")" ;

```

```

type3 --> not. type3 ;
type3 --> basic-type ;
not --> "~" ;
or --> "∨" ;
imply --> "⊃" ;
basic-type --> "P" ; "⊥" ;

```

Syntax of meta language :

```

term1 --> meta-term, "(", meta-var, ")" ;
term2 --> meta-term ;
meta-term --> "F" ;
type3 --> meta-type ;
meta-type --> "A" ; "B".

```

Note that the syntax definition for meta language is needed for defining inference rules schematically.

Inference Rules

Intuitionistic type theory is defined by a number of natural deduction style inference rules [Martin-Löf 84]. For our purpose of illustration we consider just four rules and one rewrite rule. These are the rules for function introduction and elimination, the two rules for \vee introduction, and the rewrite rule in lieu of the definition $\neg A = A \supset \perp$.

$$\begin{array}{c}
 \begin{array}{c} [x \in A] \\ : \\ F(x) \in B \end{array} \\
 \hline
 \lambda x. (x) \in A \supset B \qquad (\lambda\text{-introduction } (\lambda\text{-I}))
 \end{array}$$

$$\begin{array}{c}
 a \in A \quad f \in A \supset B \\
 \hline
 f \bullet a \in B \qquad (\supset\text{-elimination } (\supset\text{-E}))
 \end{array}$$

$$\begin{array}{c}
 a \in A \\
 \hline
 \text{inl}(a) \in A \vee B \qquad (\text{inl-introduction } (\text{inl-I}))
 \end{array}$$

$$\begin{array}{c}
 b \in B \\
 \hline
 \text{inr}(b) \in A \vee B \qquad (\text{inr-introduction } (\text{inr-I}))
 \end{array}$$

$$\frac{A = \perp}{\sim A} \quad (\text{definition as rewrite rule})$$

Proof

The manual proof of a tree form is described below and the overall screen layout is shown in Fig. 3

$$\begin{array}{c}
\begin{array}{c}
\text{[x} \in \text{P]} \\
\hline
\text{inl-I} \\
\text{inl(x)} \in \text{P} \vee (\text{P} \supset \perp) \\
\hline
\text{[f} \in (\text{P} \vee (\text{P} \supset \perp)) \supset \perp] \quad \supset\text{-E}
\end{array} \\
\hline
\begin{array}{c}
\text{f} \bullet \text{inl(x)} \in \perp \\
\hline
\lambda\text{-I} \\
\lambda\text{x. f} \bullet \text{inl(x)} \in \text{P} \supset \perp \\
\hline
\text{inr-I} \\
\text{inr}(\lambda\text{x. f} \bullet \text{inl(x)}) \in \text{P} \vee (\text{P} \supset \perp) \quad \text{[f} \in (\text{P} \vee (\text{P} \supset \perp)) \supset \perp] \\
\hline
\supset\text{-E} \\
\text{f} \bullet \text{inr}(\lambda\text{x. f} \bullet \text{inl(x)}) \in \perp \\
\hline
\lambda\text{-I} \\
\lambda\text{f. f} \bullet \text{inr}(\lambda\text{x. f} \bullet \text{inl(x)}) \in (\text{P} \vee (\text{P} \supset \perp) \supset \perp) \supset \perp \\
\hline
\text{def} \\
\lambda\text{f. f} \bullet \text{inr}(\lambda\text{x. f} \bullet \text{inl(x)}) \in \sim\sim(\text{P} \vee \sim\text{P})
\end{array}
\end{array}$$

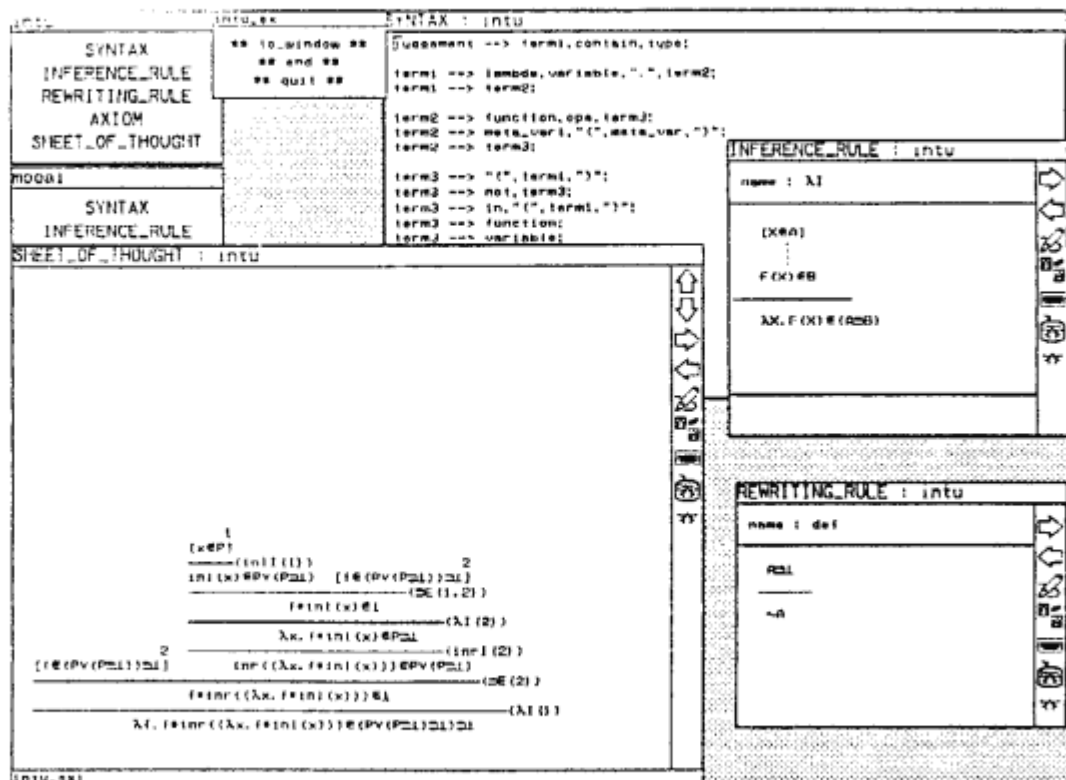


Fig. 3 Screen layout of a constructive proof

Hoare logic for program verification

Hoare logic introduced by Hoare [Hoare 69] is the most well known logic for the axiomatic definition of programming languages and reasoning about programs. Here we exemplify how such a notationally complicated formal logic can be easily dealt with on EUODHILOS. The principal formula in Hoare logic is a form of $P\{S\}Q$, where P and Q are first-order formulas and S is a program in an ALGOL-like programming language.

Language in terms of DCG

Syntax for object language :

```
formula --> formula, "⊃", formula1 ;
formula --> formula1 ;
formula1 --> formula1, "∨", formula2;
formula1 --> formula2 ;
formula2 --> formula2, "∧", formula3 ;
formula2 --> formula3 ;
formula3 --> "(", formula, ")";
formula3 --> "~", formula3 ;
```



```

formula3 --> "true" ;
formula3 --> term, "=", term ;
h-formula --> formula, "(", program, ")" , formula ;
term --> variable ;
term --> constant ;
term --> term, "+", term ;
term --> term, "×", term ;
term --> term, "!" ;
variable --> "x" ; "y" ; "z" ; meta-var ;
constant --> "1" ; "0" ;
program --> assignment-statement ;
           program, ";", program ;
           while, formula, "do", program, "od" ;
           if, formula, "then", program, "else", program, "fi" ;
assignment-statement --> variable, ":", term ;

```

Syntax for meta language :

```

program --> meta-program ;
formula3 --> meta-formula, "(", meta-term, ")" ;
formula3 --> meta-formula ;
meta-program --> "A" ; "B" ;
meta-var --> "X" ;
term --> meta-term ;
meta-term --> "T" ;
meta-formula --> "P" ; "E" ; "F" ; "G".

```

Axioms and Theorems

- (1) $E \wedge F \supset E$ (Conjunction-elimination)
- (2) $P(X) \wedge X = T \supset P(T)$ (Substitution)
- (3) $P(T) \{X := T\} P(X)$ (Assignment axiom)
- (4) $\text{true} \supset 1 = 0!$ (Arithmetic)

Rewriting rule

$$\begin{array}{c}
 z = y! \\
 \hline
 z \times (y + 1) = (y + 1)! \quad \text{(Arithmetic rule)}
 \end{array}$$

Inference Rules

$$\frac{E \supset F \quad F(A)G}{E(A)G} \quad (\text{Consequence rule 1})$$

$$\frac{E(A)F \quad F \supset G}{E(A)G} \quad (\text{Consequence rule 2})$$

$$\frac{E(A)F \quad F(B)G}{E(A; B)G} \quad (\text{Composition rule})$$

$$\frac{E \wedge F(A)G \quad E \wedge \sim F(B)G}{E(\text{if } F \text{ then } A \text{ else } B \text{ fi})G} \quad (\text{Conditional rule})$$

$$\frac{F \wedge G(A)F}{F(\text{while } G \text{ do } A \text{ od})F \wedge \sim G} \quad (\text{Repetition rule})$$

In Fig. 4 we show the screen layout of the correctness proof of a factorial program with the precondition "true" and postcondition $z = x!$.

3. $\exists w(C(w) \ \& \ \neg \forall y(C(y) \ \& \ H(y,y) \supset H(w,y,y) \ \& \ O(w,g)) \ \& (C(y) \ \& \ \neg H(y,y) \supset H(w,y,y) \ \& \ O(w,b)))) \supset \exists v(C(v) \ \& \ \forall y((C(y) \ \& \ H(y,y) \supset H(v,y) \ \& \ O(v,g)) \ \& (C(y) \ \& \ \neg H(y,y) \supset H(v,y) \ \& \ O(v,b))))$
4. $\exists v(C(v) \ \& \ \forall y((C(y) \ \& \ H(y,y) \supset H(v,y) \ \& \ O(v,g)) \ \& (C(y) \ \& \ \neg H(y,y) \supset H(v,y) \ \& \ O(v,b)))) \supset \exists u(C(u) \ \& \ \forall y((C(y) \ \& \ H(y,y) \supset \neg H(u,y)) \ \& (C(y) \ \& \ \neg H(y,y) \supset H(u,y) \ \& \ O(u,b))))$

Conclusion :

$$\vdash \neg \exists x(A(x) \ \& \ \forall y(C(y) \supset \forall zD(x,y,z)))$$

(no algorithm to solve the halting problem exists)

(3) Proof by induction [Eriksson 82]

$$\vdash \forall x. \text{append}(\text{append}(x,y),z) = \text{append}(x,\text{append}(y,z))$$

(associativity of append function)

Propositional modal logic (T)

$$\vdash \langle \rangle p \wedge [](p \supset q) \supset \langle \rangle (p \wedge q)$$

(A strong correctness assertion is implied from a termination assertion and a weak correctness assertion

[Burstall 74])

Intensional logic and reflective proof

$$\vdash P:t \Rightarrow \vdash \forall x:a. P:t \quad \text{(Generalization rule) [Gallin 75]}$$

In the future we plan to attack logic of knowledge and belief, various other logics of programs such as dynamic logic, non-monotonic logic, relevant logic and so on.

RELATED WORKS

Much work has been devoted to building systems for checking and building formal proofs in various logical systems. A number of ways which may be used for assisting human reasoning, including automatic theorem proving, proof checker [de Bruijn 80][Weyhrauch 80][Ketonen 84][Sawamura 86], proof constructor [Gordon 79][Constable 82][Constable 86] and general system for computer-aided reasoning [Coquand 85][Sawamura 87][Griffin 87][Harper 87][Minami 88][Sato 88a], are comparatively examined in [Minami 88]. Here we are confined ourselves to various approaches to general system for computer-assisted reasoning to which much attention have been recently paid. Let us first briefly see the distinction of formal system description language in each approach.

In [Sawamura 86], Prolog is employed as logic description language as well as an implementation language of proof constructor. In [Felty 88] and [Miller 87], λ Prolog, which is

a higher-order version of Prolog and hence more expressive than Prolog, is proposed to specify theorem provers. In [Harper 87] and [Griffin 87], Martin-Löf's intuitionistic type theory is applied for building a logical framework (LF) which allows for a general treatment of syntax, inference rules, and proofs in terms of a typed λ -calculus with dependent types. It also has the advantage of a smooth treatment of discharge and variable occurrence conditions in rules. In [Reps 84], the axioms and inference rules of a formal logical system can be expressed as productions and semantic equations of an attribute grammar. Then, dependencies among attributes, as defined in the semantic equations of such a grammar, express dependencies among parts of a proof. In [Gordon 82], the metalanguage (ML) for interactive proof in LCF [Gordon 79], a polymorphically typed, functional programming language, are used to show how logical calculi can be represented and manipulated within it. In [Abrial 84], constructing a general-purpose proof checker is undertaken through devising a theory of proofs. It is "general purpose" in that it may take as input the axiomatization of a formal theory together with a proof written within this theory. A theory of proofs is a kind of a specification language for formal system from the viewpoint of software engineering, and also a formal system description language. His approach is based on the rigorous approach to program construction : to define a theory and then to apply it.

In addition to such a purely theoretical interest as what a general theory of logics is, an important benefit of these treatments of formal systems is, although their approaches are different, that logic-independent tools for proof editors, proof checkers, and proof constructors can be constructed. For logic-dependent tools, we think that it would be better to provide them by designing an appropriate metalanguage such as ML [Gordon 79].

Among these general-purpose reasoning assistant systems, it can be safely said at this stage that it is only our system EUODHILOS that incorporates such a distinctive feature as providing methodology plus logic defining capability.

CONCLUDING REMARKS AND FUTURE RESEARCH DIRECTIONS

We have presented the unique features of a general-purpose reasoning assistant system EUODHILOS which is under development. And also we have shown its advantages of our approach and potentials through a number of formal systems and their proof examples. As a matter of fact, we have been confirmed in the following intended points by those experiments.

(i) Advantages of generality

The generality of EUODHILOS have been tested by using it to define various logics and to verify proofs expressed within them. Almost every logic together with its proof example was created in several hours. If we had have to develop a reasoning system with same functions as EUODHILOS for each logic from scratch, how much time would it have taken to do it ? And we would have to do another job in the almost same manner if we had needed other logic.

Therefore we may conclude that EUODHILOS have demonstrated the usefulness of its generality in a much wider range of applications.

(ii) Definite clause grammar approach to the definition of logical syntax

Definite clause grammar is more natural and easier for users to define a logical syntax, compared with other approaches to logical system description languages. And also it requires less knowledge to describe. A wff editor and a facility to test the defined language serve to check the intended syntax. These greatly lighten a user's burden in setting up his own language.

(iii) Proving methodology based on sheets of thought

(iv) Visual interface

An attempt of constructing a general-purpose reasoning assistant system is, however, only at the initial stage of research and development, and is lacking a number of significant issues which should be taken into consideration. We shall touch upon some of future research themes which may be helpful to augment and deepen the current level of EUODHILOS.

(1) Augmentation of formal system description language

The current state of the formal system description language is deficient in some respects. Much efforts have to be paid on making the logic description language more expressive. For example, in the current framework, rule descriptions of Gentzen's sequent calculus, tableaux method, some formulation of relevant logic, etc., seem not to be expressible. Furthermore, automatic mechanism checking rule application conditions are not incorporated in EUODHILOS. To overcome these deficiencies, we would need some more flexible rule description language and method.

(2) Investigation of higher-level supporting functions for reasoning

Developing a language for proof strategies, incorporating metatheory, etc., are important subjects since these could attain increasing the naturalness and efficiency of proofs.

(3) Maintaining a relational dependency among various theories

Various theories or logics are used to be involved in a proof. Let us consider the following situation : There exists a number of theories or logics together with various kinds of databases, they may be mutually dependent in the sense of the referential relations and we want to modify or revise a theory or underlying logic. Then what happens? Obviously, relational inconsistencies among theories may arise with such a modification and revision of theories or logics. The reader will notice that this is a kind of non-monotonic phenomenon.

(4) Opening up a new application field of reasoning by EUODHILOS

The unique features and potentials of EUODHILOS could suggest a new direction to CAI system for logics. We are now particularly interested in clarifying the feasibility of using EUODHILOS as a tool of logical model construction and specialized use of EUODHILOS such as a basis of computer-aided programming.

(5) Improvement and refinement of human-computer interface for the reasoning system

We have tried to analyze intrinsically how reasoning-oriented human-computer interface should be. However, it seems to lack a uniform and systematic point of view for such an interface.

ACKNOWLEDGEMENTS

The authors are grateful to Mr. T. Hai of FUJITSU LABORATORIES LTD. for his helpful discussion and cooperation in developing EUODHILOS. This work is part of a major research and development of the Fifth generation computer project conducted under a program set up by the MITI.

REFERENCES

- [Abrial 84] Abrial J. A. : The mathematical construction of a program, Science of Computer Programming, Vol. 4, pp. 45-86, 1984.
- [Backhouse 88] Backhouse, R. and Chisholm, P. : Do-it-yourself type theory (Part 1), Bull. of EATCS, No. 34, pp. 68-110, (Part 2), *ibid.*, No. 35, pp. 205-245, 1988.
- [Burkholder 87] Burkholder, L. : The halting problem, SIGACT NEWS, Vol. 18, No. 3, pp. 48-60, 1987.
- [Burstall 74] Burstall, R. : Program proving as hand simulation with a little induction, Proc. of IFIP Congress 74, North-Holland Pub. Co., pp. 308-312, 1974.
- [de Bruijn 80] de Bruijn, N. G. : A survey of the project automath, in: Seldin and Hindley (eds.), TO H. B. Curry : Essays on Combinatory logic, Lambda calculus and Formalism, Academic Press, pp. 579-606, 1980.
- [Coquand 85] Coquand, T. and Huet, G. : Constructions : A higher order proof system for mechanizing mathematics, LNCS 203, pp. 151-184, 1985.
- [Constable 82] Constable, R. L., Johnson, S. D. and Eichenlaub, C. D. : An introduction to the PL/CV2 programming logics, LNCS, Vol. 135, Springer, 1982.
- [Constable 86] Constable, R.L., et al. : Implementing mathematics with the Nuprl proof development system, Prentice-Hall, 1986.
- [Eriksson 82] Eriksson, A., Johansson, A. -L. and Tärnlund, S. -A. : Towards a derivation editor, Proc. of the 1st Int. Logic Programming Conf., 1982.
- [Felty 88] Felty, A. and Miller, D. : Specifying theorem provers in a higher-order logic programming language, LNCS, Vol. 310, pp. 61-80, 1988.
- [Gallin 75] Gallin, D. : Intensional and higher-order modal logic, with applications to Montague semantics, North-Holland, 1975.
- [Genesereth 87] Genesereth, M. R. and Nilsson, N. J. : Logical foundation of artificial intelligence, Morgan Kaufmann, 1987.
- [Gordon 79] Gordon, M. J., Milner, A. J. and Wadsworth, C. P. : Edinburgh LCF, LNCS, Vol. 78, Springer, 1979.
- [Gordon 82] Gordon, M. J. C. : Representing a logic in the LCF metalanguage, in: D. Neel (ed.), Tools and notions for program construction, Cambridge U. P., pp. 163-185, 1982.
- [Goguen 83] Goguen, J. A. and Burstall, R. M. : Introducing institutions, LNCS, Vol. 164, Springer, pp. 221-270, 1983.

- [Griffin 87] Griffin, T. G. : An environment for formal system, ECS-LFCS-87-34, Univ. of Edinburgh, 1987.
- [Harner 87] Harper, R., Honsell, F. and Plotkin, G.: A framework for defining logics, Proc. of Symposium on Logic in Computer Science, pp. 194-204, 1987.
- [Hoare 69] Hoare, C. A. R. : An axiomatic basis for computer programming, CACM, Vol. 12, No. 10, pp. 576-580, 583, 1969.
- [Ketonen 84] Ketonen, J. and Weening, J. S. : EKL - An interactive proof checker, User's reference manual, Dept. of Computer Science, Stanford Univ., 1984.
- [Kitagawa 63] Kitagawa, T. : The relativistic logic of mutual specification in statistics, Mem. Fac. Sci. Kyushu Univ. Ser. A. Vol. 17, No. 1, 1963.
- [Kunst 76] Kunst, J. : Making sense in music I - The use of mathematical logic, Interface 5, pp. 3-68, 1976.
- [Lakatos 76] Lakatos, I. : Proof and refutations - The logic of mathematical discovery, Cambridge Univ. Press, 1976.
- [Langer 25] Langer, S. K. : A set of postulates for the logical structure of music, Monist 39, pp. 561-570, 1925.
- [Martin-Löf 84] Martin-Löf, P. : Intuitionistic type theory, Bibliopolis, 1984.
- [Matsumoto 83] Matsumoto, Y., Tanaka, H., Hirakawa, Miyoshi, H. and Yasukawa, H. : BUP: A bottom-up parser embedded in Prolog, New Generation Computing, Vol. 1, pp. 145-158, 1983.
- [Miller 87] Miller, D. and Nadathur, G. : A logic programming approach to manipulating formulas and programs, Proc. of IEEE Symposium on Logic Programming, pp. 380-388, 1987.
- [Miller 87] Miller, D., Nadathur, G. and Scedrov, A. : Hereditary Harrop formulas and uniform proof systems, Proc. of Symposium on Logic in Computer Science, pp. 98-105, 1987.
- [Minami 88] Minami, T., Sawamura, H., Sato, K. and Tsuchiya, K. : EUODHILOS : A general purpose reasoning assistant system - concept and implementation -, to appear in LNCS, 1988.
- [Paulson 86] Paulson, L. C. : Natural deduction as higher-order resolution, J. Logic Programming, Vol. 3, pp. 237-258, 1986.
- [Peirce 74] Peirce, C. S. : Collected Papers of C. S. Peirce, Ch. Hartshorne and P. Weiss (eds.), Harvard Univ. Press, 1974.
- [Pereira 80] Pereira, F. C. N. and Warren, D. H. D. : Definite clause grammars for language analysis - A survey of the formalism and a comparison with augmented transition networks, Artificial Intelligence, Vol. 13, pp. 231-278, 1980.
- [Rahn 79] Rahn, J. : Logic, set theory, music theory, College Music Symposium, 19(1), pp. 114-127, 1979.
- [Reps 84] Reps, T. and Alpern, B. : Interactive proof checking, ACM Symp. on Principles of Programming Languages, pp. 36-45, 1984.
- [Sato 88a] Sato, K., Tsuchiya, K., Sawamura, H. and Minami, T. : General-purpose reasoning assistant system EUODHILOS - its unique functions and implementation -, 1988. (in preparation)
- [Sato 88b] Sato, K. and Tsuchiya, K. : EUODHILOS reference manual, Fujitsu Ltd., 1988. (in Japanese)
- [Sawamura 86] Sawamura, H. : A proof constructor for intensional logic, with S5 decision procedure, IIAS R. R., No. 65, 1986.
- [Sawamura 87] Sawamura, H. and Minami, T. : Conception of general-purpose reasoning assistant system and its realization method, 87-SF-22, WGFS, IPS, 1987. (In Japanese).

[Smets 88] Smets, P., Mamdani, A., Dubois, D. and Prade, H. : Non-standard logics for automated reasoning, Academic Press, 1988.

[Smullyan 85] Smullyan, R. : To mock a mockingbird, and other logical puzzles including an amazing adventure in combinatory logic, Alfred A. Knopf, Inc., 1985.

[Weyhrauch 80] Weyhrauch, R. W. : Prolegomena to a theory of mechanized formal reasoning, Artificial Intelligence, Vol. 13, pp. 133-179, 1980.