Vol. 36 No. 3 Transactions of Information Processing Society of Japan Mar. 1995

Regular Paper

A General-Purpose Reasoning Assistant System EUODHILOS

— Basic Features and Potential Usefulness —*

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

HAJIME SAWAMURA,† TOSHIRO MINAMI,† KAORU YOKOTA††
and KYOKO OHASHI††

Much work has been devoted to special-purpose reasoning assistant systems whose underlying logics are fixed. In contrast to 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 or her own logical system relevant for the intended problem domain and to reason about it. In the first half of the paper, the need, significance and design principle of EUODHILOS: a general-purpose system for computer-assisted reasoning, is discussed, then the system overview is described, placing emphases on the following three points: (1) an expressive and tractable framework for representing a logic, (2) a powerful and flexible proof construction facility, and (3) a visual reasoning-oriented human-computer interface for ease of use. In the latter half, the potential 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.

1. Introduction

A new dimension of computer-assisted reasoning research is 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 construct proofs in the defined system. We have named this system EUODHILOS, an acronym reflecting our philosophy or observation that *every universe of discourse has its logical structure.*

In these days, various logics play important and even essential roles in computer science and artificial intelligence (e. g., Ref. 41), 42)), and surprisingly in aesthetics which has been thought of as being in a directly opposite position to logic (e. g., Ref. 19), 20)), as well as in other scientific theories (e. g., Ref. 4), 27), 45)). Specifically, it can be said that logics 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, people 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.), inference rules, rewriting rules, proofs, proof strategies, definitions and so on, depending on each logic under consideration. Thus, it is desirable to find a general theory of logics and a general-purpose reasoning assistant system that captures the uniformities and idiosyncrasies of a large class of logics so that much of this effort can be expended once only. A similar observation and motivation can be found in the papers of Ref. 14) and 15), although the approaches differ. In this paper, we aim at building a general and easy to use system which handles as many logics as possible and allows us to reason in various ways[25),35)].

* This paper is a revised version of the paper presented at the Seventh International Conference on Logic Programming, 1990, Jerusalem.

† Institute for Social Information Science, FUJITSU LABORATORIES LTD., 140 Miyamoto, Numazu, Shizuoka 410-03, JAPAN
†† Software Laboratory, FUJITSU LABORATORIES LTD., 1015 Kamikodanaka, Nakahara-ku, Kawasaki, Kanagawa 211, JAPAN

There are three major subjects to be pursued for such an interactive and general reasoning support system. One is a framework 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 in the conception of ideas in reasoning, and in their further promotion.

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 according to the respective problem domains.

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 is described. We emphasize the following three points: (1) an expressive and tractable framework for representing a logic, (2) a powerful and flexible proof construction facility, and (3) a visual reasoning-oriented human-computer interface for ease of use. In the latter half of the paper, the potential and usefulness of EUODHILOS are shown through experiments and experiences of its use by a number of logics and proof examples therein. These have been used or devised in computer science, artificial intelligence and other related fields.

2. Need, Significance and Design Philosophy

Much work has been devoted to special-purpose reasoning assistant systems whose underlying logics are fixed (e.g., Ref. 5), 7), 12), 18), 43), 44)). However, we are exploring a new dimension in a general-purpose reasoning assistant system.

We first take up some issues concerned with the generality in reasoning assistant system and several aspects of viewing such a generality. We have already found and recognized that in these days a logic or logical methodology forms a kind of paradigm for promoting computer science, artificial intelligence and so on. 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 and idiosyncrasies of a large class of logics so that much effort for providing reasoning facilities can be expended once only 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 a rigorous approach to program construction. Abrial[1] claims that a general-purpose proof checker could 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. Note that every program (universe) to be constructed (studied) has its underlying theory (logical structure). The third issue concerns the construction of a logical model, or more generally methodology of science. We observe that human reasoning process consists 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 phase (1) could be aided mechanically since some part of phase (1) is very creative. On the other hand, it is very likely that phases (2) and (3) could be largely supported mechanically by allowing the modification or revision of 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. Philosophical aspects of the generality from a logical point of view can be found in Ref. 20) and 10). A logic is, in a broad sense, a way of doing things. In this sense it is not a surprising fact that there may exist a number of logics for things. Also it is well known that a logic has various styles in its formulation such as Gentzen's LK, NK, Hilbert's linear style, etc., and that these are mathematically equivalent. However, if a logical system is to be viewed as a form of representation of a

system of self-consciousness, then we will have to think of these various logic formulations as different[10].

All this discussion may be summarized as, to borrow Langer's statement[20], "*Every universe of discourse has its logical structure*". Thus it eventually supports our discussions about the need and significance of the generality in reasoning assistant system from the philosophical point of view.

The above discussion led us to the research and development of general-purpose reasoning assistant system EUODHILOS with the following outstanding features:

- Use of the definite clause grammar (DCG) in representing a logic
- Proving methodology using sheets of thought
- Reasoning-oriented human-computer interface.

In what follows, we will sketch each of these features in more detail.

3. An Overview of EUODHILOS

We list the main functional features of EUODHILOS and explain them briefly. We start by describing the language of a logic to EUODHILOS. Fundamentally, EUODHILOS has almost no defaults except some logically proper conventions in representing a logic.

3.1 Formal System Description Language

What is a logic? What language should be expressive enough to describe or deal with logics? The answers to these questions could turn out to define the formal system description language for capturing the uniformities and idiosyncrasies 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 the formal system description language. In this, these attempts have shared the goal of EUODHILOS, e.g., Prolog is employed as a logic description language in Ref. 34), λProlog in Ref. 9) and 24), typed λ-calculus with dependent types in Ref. 14) and 15), a specification language for a wide variety of logics in Ref. 1), an attribute grammar formalism in Ref. 32), a metalanguage ML in Ref. 13) and a higher-order logic in Ref. 28).

Almost all of contemporary logics may be considered as having a logical framework consisting of a proof theory and a model theory. A proof theory specifies the syntactical part of a logic and a model theory specifies the semantical part of a logic. In this paper we are mainly concerned with specifying the syntactical aspect of a logic. The syntax of a formal system is made up of two constituents: language system and derivation system.

(1) The language system

A language is a tool for talking about objects and is formed from underlying primitive symbols. A logical language is one in which propositions are expressed and reasoned about. It is usually specified by utilizing some of the following: variables, constants and functions as individual symbols, predicates (including equality), logical connectives, auxiliary symbols, etc. Attributes such as type, sort, arity, operator precedence are sometimes associated with some of these symbols. Once these primitive symbols are specified, complexities such as terms, formulas, etc., are constructed from them by formation rules. Also, notational conventions for defining or abbreviating symbols may be required. At this point, we face our next fundamental question: what kind of metalanguage is natural and sufficient to describe such an object language?

(2) The derivation system

The derivation system gives us a means of manipulating a logical language. It is specified by axioms, inference rules, derived rules, rewriting rules, and concepts of proofs, etc. Insofar as we confine ourself to the existing types of formal systems, we can enumerate primitive operations. Included in these are substitution, replacement, juxtaposition, detachment, renaming, unification and instantiation. These are common operations within various logics except for the differences of languages. Since we are considering a general-purpose reasoning system for logics, we have to provide a general method for those symbol manipulations. So, our next fundamental question is: what sort of primitive operations and constraints on objects are sufficient to manipulate logics and how could these be provided in a generic manner?

In addition to these questions, we need to pay attention to the concepts "free", "bound" and "something is free for a variable in an expression".

In what follows, we will attempt to answer these fundamental questions.

3.2 Specifying a Logical Syntax and the Expressiveness of the Definite Clause Grammar

In EUODHILOS, an object language to be

used is designed and defined by a user. The meta language is definable also. This is indispensable for the schematic specifications of axioms, inference rules and rewriting rules and schematic proofs. A current solution for formal system description language is to employ so called definite clause grammar formalism (DCG)[30], where the problem of recognizing or parsing a string of a language is transformed into a problem with a proof 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 make CFG so important for language theory such as the modularity of a grammar description and the recursive embedding of phrases which are characteristic of almost all interesting languages, including the 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 areas[30]: (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 a semantical facility with grammar rules[32]. The necessity for context-dependency is often encountered in defining logical syntax (see an example in intensional logic below).

Let us describe some concrete examples of the syntax definition in order to see the paradigm of definite clause grammar formalism. 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

```
term(f(T,S))-->
    functor2,"(",term(T),",",term(S),")";
functor2-->"f".
```

The defining clause of terms in the intensional logic[11] such as "If A is a term of type (a, b) and B a term of type a, then A∘B is a term of type b" is represented as

```
term(A∘B, b) -->
    term(A,(a,b)),"∘", term(B, a).
```

However, we found that it is not a good way to force users to specify the internal structures of expressions manipulated by a computer. This is necessary as well for the uniform treatment of various logical grammars. In EUODHILOS, the internal structures of expressions will be automatically generated from the definite clause grammar rules (see below). Thus, the above rules can be simply given to EUODHILOS as follows:

```
term-->functor2, "(", term, ",", term, ")";
functor2--> "f"., and
term(b)-->term((a,b)), "∘", term(a).,
```

respectively.

Furthermore, the definite clause grammar formalism is slightly augmented with operator precedence and special constructs to handle proper logical concepts such as variable binding, scope, substitution/variable occurrences, schema variables. In EUODHILOS, the special built-in construct "bind-op" deals with the concept of a variable being bound over the scope of a sub-expression. For example, in the following two clauses, the nonterminal "variable" denotes a bound variable and its scope is over "formula".

```
formula-->bind_op, variable, formula;
bind_op-->"∀"|"∃"|"λ";
```

The expression like "P[X]" represents the variable "X" possibly occurs in an expression P, and the expressions "P[e1/e2]" represents the results obtained by replacing one or more occurrences of e2 in P by e1. Then, either the template to show which occurrences are to be replaced has to be given by the user, or EUODHILOS automatically generates the possible templates in context, later chosen by the user. In this manner, EUODHILOS handles substitution (or replacement) problems which generally fall into one of the following categories; partial substitution like substituitivity of equals (e.g., $P(a),\ a=b \vdash P(b/a)$) and total substitution like specialization (e.g., $\forall x.P(x) \vdash P(a/x)$ provided a is free for x in P). The special nonterminal with the prefix "meta" is used to define meta/schema variables in the definite clause grammar.

3.3 Automatic Generation of a Parser and an Unparser

Once a definite clause grammar definition for a logical syntax has been given, it is first converted to the definite clause grammar associated with the internal structures of expres-

sions. The conversion is done with the help of an operator declaration provided by a user, which is for indicating which syntactical element should be viewed as an operator in the grammar rule. Then the bottom-up parser for the new grammar is automatically generated, employing the BUP generation method for the definite clause grammar[22]. The reason why we do not generate a top-down parser for the defined language is to avoid the anomaly of left-recursiveness which often appears in the ordinary definition of a logical syntax. The automatic method for generating the internal structures of the expressions of a language have been provided by us[26]. The unparser (or generator) for the internal structures is also automatically constructed with the help of the operator precedence declaration provided by a user. The generated parser and unparser are internally used in all the succeeding phases of symbol manipulations.

It is clear that our approach based on DCG is far superior to the other approaches based on attribute grammar (e.g., Ref. 32)), in which we have to provide the internal and external representations of expressions, and hence those automatic generations of a parser, an unparser and internal structures greatly lighten a user's burden in setting up his or her own language and taking care of it. Readers interested in the details of the algorithms can find these in Ref. 26).

3.4 Specifying a Derivation System

A derivation system consists of axioms, inference rules and rewriting rules. Axioms are simply presented in a list of formulas. For the specification of rules, there are two important issues to be considered: side conditions and dependency.

An inference rule is specified in natural deduction style[31] in three parts: the derivations of the premises of the rule, the conclusion of the rule, and the restrictions that are imposed on the derivations of the premises and the conclusion, such as variable occurrence conditions and substitution conditions. Actually, inference rules are presented schematically in terms of meta/schema variables as follows:

$$\frac{\begin{matrix}[\text{Assumption}_1] & [\text{Assumption}_2] & \cdots & [\text{Assumption}_n] \\ \vdots & \vdots & & \vdots \\ \text{Premise}_1 & \text{Premise}_2 & \cdots & \text{Premise}_n\end{matrix}}{\text{Conclusion}}$$

where brackets are used to encompass a temporary 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. In forward reasoning, an inference rule may be permitted to apply if all the premises are obtained in this manner and the application condition is satisfied. In backward reasoning, discharging assumptions, generating some assumptions and checking the application conditions are in general impossible and hence delayed until completing the partial proof tree under construction.

In our approach, the side conditions of a rule are supposed to be described in terms of the built-in primitive side conditions and their combinations. Schematically, among those primitive conditions are

(a) *t* is free for *x* in *P*,
(b) *x* is not free in *P*, and
(c) *a* is an eigenvariable,

where the expressions like *t* are placeholders to be substituted for. Then EUODHILOS can check those side conditions automatically in the proof process. For other side conditions which can not be handled in this way, we have provided the interface with a user-programmed side conditions checker for EUODHILOS.

The dependency of a conclusion on temporary assumptions is automatically calculated by the ordinary method for the natural deduction[18),31]. Other dependency calculation also can be specified in EUODHILOS if we specify it by using both an idea of dependency as a tag/label and rewriting rules. For example, let us consider the $\wedge$-introduction rule of some relevant logic,

$$\frac{A^{\alpha}\ B^{\alpha}}{A \wedge B^{\alpha}}$$

where the superscript α denotes dependency on which the formula depends. The rule says that we can infer the formula $A \wedge B$ with dependency α if we have A and B with the same dependency α. Such a rule may be very natu-

rally specified within the rule description convention of EUODHILOS by incorporating dependency into an object formula, as follows:

$$\frac{\alpha => A \quad \alpha => B}{\alpha => A \wedge B}$$

Then some operations on dependency α may be needed, but they are easily describable as rewriting rules (see the subsection 4.9).

Defining the derived rules is allowed if they are justified for validity on a sheet of thought described below. The derived rules would become useful when we wish to shorten the lengthy and tedious derivation steps.

Rewriting rules are useful for handling equational reasoning often appearing in ordinary mathematical practice. A rewriting rule is specified with a pair of forms before and after rewriting in the following schematic format:

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

The rule is applied to an expression when it has a subexpression which matches to the exp_1, and the resulting expression is obtained by replacing the subexpression with the appropriate expression of the exp_2. EUODHILOS automatically generates many possible forms of an expression which may be obtained by successive applications of a given rewriting rule. Users can then choose an intended one from them.

Many well-known styles of logical stipulation can be treated within this framework—for example, Hilbert's, Gentzen's, equational, and even tableau styles.

3.5 Proof Construction 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 required in reasoning. Furthermore, performing formal derivations is time-consuming and error-prone. Readers may notice that such a situation is quite similar to 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 can overcome the problems with errors and the time-consuming task. The current version of EUODHILOS has the following unique facilities which are able to support natural and efficient constructions of proofs in the defined formal system.

(1) Sheets of thought

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[29]. He actually developed an extensive diagrammatic calculus which he intended as a general reasoning tool. A sheet of thought, in our case, is a field of thought where we are allowed to draft a proof, to compose proof fragments or detach a proof, to reason using lemmas, etc., while a sheet of assertion is a field of thought where an existential graph as an icon of thought is supposed to be drawn. Proving by the use of sheets of thought turns out to yield proof modularization which is considered important particularly for proving in the large. It may be beneficial to note that proof modularization is approximately equal to the concept of program modularization, to borrow the term of software engineering.

(2) Tree-form proof

As mentioned above, inference and rewriting rules are presented in a natural deduction style. This naturally induces the construction of a proof into a tree-form proof with a justification for each line (node) indicated in the right margin. For example, the justification ($\wedge$I {1, 2}) of the following proof tree says that the conclusion B is obtained by the rule named $\supset$I from the two premises A and A$\supset$B, and it depends on the assumptions named 1 and 2 above the premises.

$$\frac{\begin{matrix}\vdots & \vdots \\ A & A \supset B\end{matrix}}{B}(\supset I\{1, 2\})$$

Consequently it leads to the explicit representation of a proof structure, in other words, proof visualization.

(3) Schema (meta) variables

The Schema variables are useful not only for the schematic specifications of axioms, inference rules or rewriting rules, but also for schematic proofs. EUODHILOS is supposed to make the meta and object distinction at the time of language definition. Then substitution and unification viewed as the common and primitive symbol operations are supposed to operate on schema variables, in addition to the usual variables.

3.6 Proving Methodology

In EUODHILOS, a proof is to be constructed interactively and the human reasoner retains the initiatives in the proof process with the facilities playing the careful assistant role with responsibility for confirming the viability of each proof step.

The predominant style of interactive reason-

ing is goal-directed, in other words, top-down or backward reasoning, whereby the user breaks a goal into subgoals. It is, however, desirable that reasoning or proof construction can be done along the natural way of thinking for human reasoners. Therefore EUODHILOS supports the other typical methods for reasoning as well. They include bottom-up reasoning (forward reasoning), reasoning in a mixture of top-down and bottom-up, reasoning by using lemma, schematic reasoning, etc. These are accomplished interactively on several sheets of thought. Below, we will describe various proof methods of EUODHILOS in detail.

(1) Input of logical expressions

Derivations begin with giving any of assumptions, premises, theorems and conclusions to sheets of thought. Axioms and theorems are inputted simply by pointing one at a time from the axiom list and theorem database respectively. Then one can expand a proof tree upward or downward by applying a rule to it. It is always possible to test whether formulas at the top of a proof tree are axioms or theorems by invoking the test command.

(2) Forward and backward reasoning

Forward reasoning is often used when we try a proof from initial formulas in a trial and error fashion. In larger proof development activities, one hopes to conquer a big and complex task by backward reasoning, dividing it into smaller and simpler ones and then putting the results together. Generally, a proof will be attained by a mixture of them—partly forward and partly backward.

In order to deduce forward by applying an inference rule, we usually start a proof by inputting formulas used as premises of the rule and in a natural deduction setting by further indicating assumptions to be discharged. Then one may select an appropriate inference rule from the rule menu which has been automatically generated at the time of logic definition, or one may input a formula as the conclusion. If one selects a rule, then the system applies the rule to the premises and assumptions, and derive the conclusion. If he/she gives the conclusion, then the system searches the rules and tries to find one which coincides with this deduction. EUODHILOS can search the candidates of applicable inference rules to the given premises as well and hence we may simply choose the intended one. In natural deduction setting of a formal system, forward reasoning may be done without inputting or indicating assumptions to be discharged. This implies that at an appropriate stage of a proof, we have to decide which assumption we should discharge. This comes from such a proper form of an inference rule that assumptions in natural deduction rules may not be necessarily used in the derivations of premises. Instead, EUODHILOS helps us doing this task in a natural way. Let us consider the proof composition from the following two proof trees.

$$\frac{\begin{array}{c}[P]^1 \cdots [P]^2\\ \vdots\end{array}}{Q}(\ldots\{1, 2\}), \quad \frac{[Q]^3}{P\supset Q}(\supset I\{3\}).$$

By simply composing them (see (5) below), we get

$$\frac{\begin{array}{c}[P]^1 \cdots [P]^2\\ \vdots\\ Q\end{array}}{P\supset Q}(\supset I\{1, 2\}).$$

Then the proposition P in the consequence is not known whether it is P^1, P^2, or any other P outside this proof tree. EUODHILOS supports the following discharging method:

(a) If the proposition P in the consequence is meant to be $P^1(P^2)$, then we choose $P^1(P^2)$ as a discharged assumption and get the new proof tree with the new justification $(\supset I\{2])$ $((\supset I\{1]))$ respectively.

(b) If the proposition P in the consequence is meant to be any other P outside the proof tree, then we may simply continue expanding the proof without any action.

In the case of backward reasoning, the reasoning process is converse to the forward reasoning, so that the intermediate proof may branch off to partially justified proof fragments and the complete justification of those partially justified proof fragments is delayed until the completion of a final proof tree.

(3) Schematic reasoning

EUODHILOS allows us to construct an abstract proof in the sense that schema/meta variables ranging over syntactic domains of an object language are permitted to occur in the process of the proof, that is, we can make a partially instantiated proof. Such schema variables are obviously very convenient for having an indeterminate or unknown predicate (such as invariant assertion in Hoare logic) unspecified temporarily in the proof constructing process.

A schematic proof, however, is not always constructed since the schema variables in the

proof may not be fully instantiated so as to promote further steps. Below we discuss how schema variables communicate with objects in EUODHILOS. Axioms are represented using schema variables, but with (possibly) some conditions on them. For example, the first-order axiom: $\forall x.(P \supset Q) \supset (P \supset \forall x.Q)$ has the condition that x is not free in P, where P and Q, and x are schema variables ranging over formulas and individual variables respectively. Note that such a condition in axioms is viewed as a side condition like those of inference rules. Thus a proof using this axiom turns out to be schematized to the extent that the schema variables P are instantiated as concrete formulas. In the case of inference rules, a proof process may be banned by its side conditions unless schema variables are enough instantiated so as to be able to check side conditions. An alternative to handle these situations would be to delay checking side conditions until schema variables are fully instantiated. To do so, every side condition which has been inherited as unchecked during the proof process would have to be kept with the final theorem which is not actually a theorem, but should be stored as a conditional theorem.

(4) Reasoning by lemmas and derived rules

Theorems constructed on the sheets and validated derived rules can be stored in the theorem database and derived rule database respectively. They are referred to and reused in the later proofs for other theorems. For large and complex proofs, derived rules are helpful for preventing proof trees from expanding more than they needs, and avoiding the repetition of the same subtrees in a proof tree. It should be noted that derived rules sometimes can play a role of so-called tactical reasoning[12] as well, although we have not yet implemented tactic and tactical reasoning which seems to be a promising way for large proof development. After using EUODHILOS systematically and over a long period of time, the theorems turn out to build up theories.

(5) Connection and separation functions on sheets of thought

(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 is viewed as one of Tarski's consequence relation common in all logics.

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

where Γ, Δ and Σ might represent sequences of formulas (possibly empty), and A and C 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 proof fragments scattered on several sheets of thought through an appropriate rule of inference. Let us take an example schema of modus ponens:

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

with the same proviso, adding that B represents a formula.

(c) Connection by unification

Two proof fragments can be connected through two unifiable formulas 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 unified to the most general proof fragment. It is, however, noted that the unification can be done through schema variables at the moment.

Besides, a connection method such as analogical matching would become extremely beneficial to intelligent reasoning system, which is left as a future subject.

(d) Separation

The separation is 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 is detached into the two fragments. Schematically, this amounts to the converse to the connection by complete matching above. In natural deducion setting of a logical system, the assumption numbers are automatically managed by the system. We will illustrate this by separating the following proof tree at the location of formula B.

$$\cfrac{\cfrac{\cfrac{[A]^1 \; [A \supset B]^2}{B}(\supset I\{1, 2\})}{C}(\ldots\{1, 2\}) \quad [D]^3}{E}(\ldots\{1, 2, 3\})$$

We then get the two proof trees on a sheet of thought as follows.

$$\frac{[A]^1 \; [A \supset B]^2}{B}(\supset I\{1, 2\}) \text{ and}$$

$$\cfrac{\cfrac{[B]^4}{C}(\ldots\{4\}) \quad [D]^4}{E}(\ldots\{4, 5\})$$

(6) Automated reasoning

In principle, there can be no mechanized way of provability except some simple logics and it will be up to the human, with the machine's help, to discover a proof. However, an interactive system like EUODHILOS depends too much on user involvement in reasoning. Some automated aspects should be incorporated in the reasoning process. Two promising ways to solve this problem may be taken into consideration. First, tactic and tactical reasoning are expected to provide flexibility in controlling the search for proofs. They also allow for blending automatic and interactive theorem proving techniques invented so far in one environment. Second, filling the gap between the scattered proof fragments on several sheets of thought would be easier than traversing a full proof search space.

For the present, we have provided for EUODHILOS an interface with automated facilities such as automated theorem provers, theorem database retriever, term rewriting systems, and so on. This can be a particularly effective way of combining and reusing tools for specific problem domains in a generic environment. A rewriting rule of EUODHILOS is semi-automated in such a way that users set the number up to which the rewriting rule is applied to an initial expression. Then EUODHILOS automatically generates many possible forms of the expression which may be obtained by successive applications of the rewriting rule.

(7) Drafting proofs

It would be quite usual to take many days for a proof to be completed, in particular for a large and complex proof. EUODHILOS has not only a theorem database but also a work area for temporarily storing scattered proof fragments on sheets of thought which have not been fully justified yet, but some of which may turn out to constitute a final proof.

3.7 Human-Computer Interface for Reasoning

In the interactive reasoning system, it is up to the user to guide the search for a proof and discover a proof with the machine's help. And the process of finding a proof is often one of trial and error, and various attempts can become very large. Therefore a good user interface should make it easy to manage proofs. In EUODHILOS the following facilities are now available as a human-computer interface for ease in communicating and reasoning with a computer, in particular facilities for inputting formulas and formula visualization.

(a) 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.

(b) 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.

(c) Stationery for reasoning

Independently of the 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.

3.8 Implementation

Exploiting the bit-map display with multi-window environment, mouse, icon, pop-up-menu, etc., EUODHILOS is implemented in ESP language (an object-oriented Prolog) on PSI-II/SIMPOS. Needless to say, Prolog serves as a good implementation language for theorem provers and interactive reasoning systems since they directly implement search and

unification which are essential operations for traversing a search space for a proof and manipulating formulas and proofs. Object-oriented facilities of ESP have played an important role in the implementation of EUODHILOS as well since it is a kind of generic or meta system in which each logic is to be constructed as an instance object of a class "logic".

In this paper, however, we will not go into the implementation issues of EUODHILOS any further. It will be described elsewhere.

4. Experiments and Experiences with EUODHILOS

We have applyied EUODHILOS to various types of reasoning[37]. Logics and proof examples that we have dealt with so far on EUODHILOS include

(1) first-order logic (NK): various pure logical formulas, the unsolvability of the halting problem and an inductive proof,
(2) second-order logic: the equivalence between the principle of mathematical induction and the principle of complete induction,
(3) propositional modal logic (T): modal reasoning about programs,
(4) intensional logic (IL)[11]: the reflective proof of a metatheorem and Montague's semantics of natural language,
(5) Martin-Löf's intuitionistic type theory[2),21)], and
(6) Hoare logic[17] and dynamic logic[16]: reasoning about program properties.
(7) General logic[40],
(8) Relevant logics[23),41)],
(9) A logic of knowledge.

Note that these logics constitute a currently well-known and wide range of logics or formal systems.

In this section, in order to demonstrate the potential and usefulness of EUODHILOS, we first show how EUODHILOS can be used to specify a logic and construct a proof under the specified logic, taking up an intuitionisic type theory. Then, we will list some other proof experiments with different logics, together with brief annotations. 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 would help to convince the readers of the potential and usefulness of EUODHILOS in a much wider range of applications. (See Ref. 37) for the detailed definition of each logic and proof examples in the experiments.)

4.1 Martin-Löf's Intuitionistic Type Theory and a Constructive Proof

The first reasoning system we have chosen as an example is a tiny subset of the intuitionistic type theory described in Ref. 21) and 2). The principal expression in the intuitionistic type theory is a judgement of the form "a∈p", reads "a is a proof of a proposition p" in formulas-as-types interpretation, where "a" is an expression in λ-calculus and "p" is a first-order formula interpreted as a type. The judgement is naturally and well described in the framework of DCG. The intuitionistic type theory is defined by a number of natural deduction style inference rules[21] which are of course best suited to our treatment of rules.

Tiny language for the type theory

The language definition basically consists of four parts: an object language, a meta language, interface between the meta and object languages and an operator definition as can be seen in **Fig. 1**.

It is noted that the syntax definition for the meta language is provided for defining inference rules schematically, and the operators have precedence in the indicated order as well as their associativity, and the functors or predicates, e. g., "inl" in the term "inl (x) ", are listed simply by themselves or the non-terminals by which they are denoted, under the heading "predicate". The operator declaration is to tell the parser that the terminal declared to be an operator or the terminal denoted by the non-terminal is entitled to become the principal operator of the internal structure for an expression generated by the grammar rule.

Inference Rules

The intuitionistic type theory is defined by a number of natural deduction style inference rules. For the 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 ∨-introduction (see **Fig. 2**), and the rewrite rule in lieu of the definition $\sim A = A \supset \bot$ (see **Fig. 3**).

We have specified both the language system and derivation system for a tiny subset of intuitionistic type theory. It should be noted that EUODHILOS allows an object logic to be represented in a way that directly reflects the proof-theoretic nature of the logic speculation

```
SYNTAX : Intuitionistic_type_theory
save  make  test  structure  print  reshape  exit

% Meta_language
meta_term --> meta_term1;
meta_type -> "A" | "B";
meta_term1 --> "F" | meta-const | meta_variable;
meta_const --> "a" | "b";
meta_variable -> "X";

% Object_language
judgement --> term,"∈ ",type;

term --> bind_op,variable,".",term|
         term,"•",term|
         "(",term,")"|
         "~",term|
         'inl","(",term,")"|"inr","(",term,")"|
         variable|constant|
         meta_term1,"(",term,")"|meta_term;

type --> type,"⊃",type|
         type,"∨",type|
         "~",type|
         "(",type,")"|
         basic_type;

variable --> "x"|"f";
constant --> "c"|"d";
basic_type --> "p"|"⊥";
bind_op --> "λ";

% Interface between meta and object languages
type --> meta_type;
variable --> meta_variable;

operator
  "~"; "∨":left; "⊃":left; "•":left; "λ"; "∈ ";
predicate
  "inl","inr", meta_term1.
```

Fig. 1 Syntax definition of intuitionistic type theory by the augmented DCG.

and in the almost same way as used in ordinary logic text books. Hence we think that our approach is much more tractable and usable to a wide class of users than other methods of logic representation[39].

We may often want to revise or modify the defined logical system, due to the inconveniences encountered later. By the inconveniences, we mean the logical system being too weak, too strong, redundant, or irrelevent. Once a logical system has been specified, the revision or modification of it is critical and must be done carefully since already established facts may not be guaranteed to hold. The current version of EUODHILOS does not warrant such a theory revision as yet. A revision, however, is safe in the case where the logical system is augmented by adding symbols, axioms and inference rules to the old system as far as the addition is consistent with the old one.

Appendix 1 displays the proof of the theorem

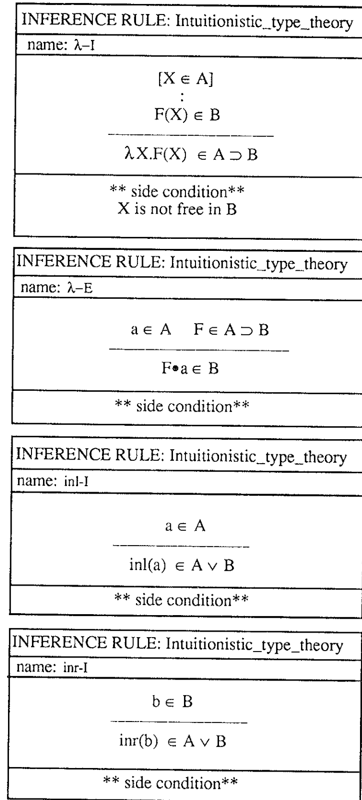


Fig. 2 Inference rules of intuitionistic type theory.

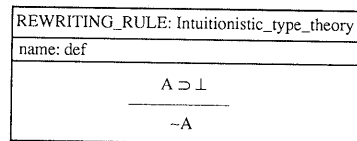


Fig. 3 Rewriting rule of intuitionistic type theory.

$\sim\sim(P \vee \sim P)$. The theorem means that the law of the double negation of the excluded middle cannot be refuted. This 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. In this paper, we will not go into the details about how the proof has been constructed using our various proof facilities and methods any further. Interested readers should refer to the paper[38].

4.2 Hoare Logic and Program Verification

Hoare logic[17] is the most well known logic for the axiomatic semantics of a programming language and the verification of a program. The principal formula in Hoare logic is a form of P{S}Q, reads "if P holds, then after execut-

ing the program S, Q holds", where P and Q are first-order formulas and S is a program in an ALGOL-like programming language. These syntactic objects are easily described in the DCG framework, as well as the inference rules of Hoare logic which is a kind of Hilbert-type logical system.

The screen layout of the proof of the following partial correctness assertion of a factorial program is shown in Appendix 2:

true {z:=1;y:=0;while∼(y=x) do
y:=y+1;z:=z*y od} z=x!

with the precondition "true" and postcondition "z=x!". For such a proof, we have often used an external theorem retriver which was connected to EUODHILOS through the theorem prover interface of EUODHILOS, in order to search for arithmetical theorems from its theorem database.

4.3 Dynamic Logic and Reasoning about Programs

Dynamic logic[16] is a kind of multi-modal logic which is an extension of classical logic. The principal formulas in dynamic logic are the dynamic formulas of the form [a]p and the dual<a>p, read informally "after executing the program a the proposition p holds", where "a" is a regular or context-free program and "p" is a first-order or dynamic formula. They can be easily dealt with in the framework of DCG. One of the example proofs in this logic is the following properties of a factorial program:

Termination:
x≧0⊃ <z:=1;
((x>0)?;z:=x×z;x:=x−1)*;
(x=0)?>true

Partial Correctness:
x=n⊃[z:=1;
((x>0)?;z:=x×z;x:=x−1)*;
(x=0)?] (z=n!)

Total Correctness:
x≧0∧x=n⊃ <z:=1;
((x>0)?;z:=x×z;x:=x−1)*;
(x=0)?> (z=n!)

4.4 Intensional Logic, Reflective Proof and Montague's Semantics

Intensional logic[11] is a higher-order modal logic based on the simple type theory, which requires context-sensitive constraints on terms. It includes a lot of complicated logical concepts which however are all well described within the framework of DCG and the rule description conventions. The following metatheorem, ⊢P: t=>⊢∀x:a. P:t (Generalization rule) is ingeniously proved using the idea of the reflection principle (Ref. 44)):

$$\frac{\text{beweis}(A)}{A}\ (\text{Reflection-1})$$

$$\frac{A}{\text{beweis}(A)}\ (\text{Reflection-2})$$

In Montague's language theory, natural language sentences are first translated into expressions in intensional logic, which in turn are analyzed by using the possible world semantics. Under the defined intensional logic, the following complicated intensional formula:

(λp:(s,(e,t)).∃x:e.(fish:(e,t)
• x:e∧p:(s,(e,t)){x:e}))
• ^λy:e.(believe:((s,t),(e,t))
• ^(walk:(e,t) • y:e) • j:e)

which is a translation of a natural language sentence "John believes that a fish walks", easily and precisely reduces to a more simple and legible one:

∃x:e.(fish:(e,t)
• x:e∧believe:((s,t),(e,t))
• ^(walk:(e,t) • x:e) • j:e).

For other logical experiments, we will merely list the typical theorems which were proved by using EUODHILOS.

4.5 First-Order Logic (with NK)

(1) Smullyan's logical puzzles (originally examples in combinatory logic)

Axioms:
1. ∀x m•x=x•x (Mockingbird condition)
2. ∀x∀y∃z∀w z•w=x•(y•w) (Composition)

Theorems:
1. ⊢∀x∃y(x • y=y) (Every bird of the forest is fond of at least one bird)
2. ⊢∃x(x • x=x) (At least one bird is egocentric or narcissistic)

(2) Unsolvability of the halting problem[6]
⊢∼∃x(A(x)&∀y(C(y)⊃∀zD(x,y,z)))
(no algorithm to solve the halting problem exists)

where the meaning of each predicate is as follows; A(x): x is an algorithm, C(y): y is a computer program in some programming language and D(x, y, z): x is able to decide whether y halts given input z.

(3) Proof by structural induction on list
⊢∀x∀y∀z.append(append(x,y),z)
=append (x, append (y, z))
(associativity of append function)

(4) Category theory

An elementary category theory have been built up on EUODHILOS, proving a number of

theorems.

4.6 Second-Order Logic and a Simple Equivalence Proof

$$\vdash \forall P[P(0) \wedge \forall n(P(n) \supset P(n+1)) \supset \forall n P(n)] \equiv \forall R[\forall n(\forall j(j<n \supset R(j)) \supset R(n)) \supset \forall n R(n)]$$

(The principle of the mathematical induction is equivalent to the principle of the complete induction.)

4.7 Propositional Modal Logic (T) and Modal Reasoning about Programs

$$\vdash <>p \wedge [\,](p \supset q) \supset <>(p \wedge q)$$

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

4.8 General Logic

General logic is a kind of Gentzen-type formal system which yields a unified account of a fairly wide range of logical systems. Diverse logics are displayed as variations on a single theme[40]. Such a general logic have been very successfully and smoothly handled on EUODHILOS by specifying those variations on a single theme as rewriting rules[36]. The proof examples in the various systems covered in Slaney's general logic include:

$p, q \vee r{:}p \;\&\; q \vee r$ (Distribution)

$X;B{:}A \;\&\; \sim A \vdash X{:}\sim B$ (Reductio ad absurdum)

true: $\exists y.(g(y) -> \forall x.g(x))$ (Baffling formula), etc.

4.9 Relevant Logic

The relevant logic we have taken is an implicational fragment of relevant logic, $R_{->}$[23]. Dependency for this logic is specified as a tag of a formula, differently from the usual set-theoretic dependency calculus, and then the tag is a composite formed from combinators satisfying some reduction rules. Tag of $R_{->}$ is to stipulate dependency of an inference so as to yield a conclusion relevantly from an antecedent. For example, the following tag rule C of $R_{->}$ can obviously be handled as a rewriting rule in EUODHILOS;

$$\frac{(tr)s=>P}{(ts)r=>P}\text{(Tag rule C : } C\alpha\beta\gamma=\alpha\gamma\beta).$$

5. Related Work

In recent years, there has been a growing interest in using computers as an aid for manipulating formal systems (e. g., Ref. 8), 14), 15) and 28)). Here, we will have to restrict ourselves to seeing only the distinction of a formal system description language in each approach since there have not yet been so much work as to the other aspects such as proving methodology for computer-assisted reasoning and reasoning-oriented human-computer interface, to such an extent that comparative studies become possible.

In Ref. 34), Prolog is employed as a logic description language as well as an implementation language of a proof constructor. In Ref. 9) and 24), λ-Prolog, which is a higher-order version of Prolog and hence more expressive than Prolog, is proposed to specify theorem provers. In Ref. 14) and 15), a typed λ-calculus with dependent types is used for building a logical framework (LF) which allows for a general treatment of syntax, inference rules, and proofs. It also has the advantage of a smooth treatment of discharge and variable occurrence conditions in rules. In Ref. 32), 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 Ref. 28), a logic is to be encoded to a subset of a higher-order logic. What they are aiming principally at seems to be automatic check of rule conditions basically in one way reasoning, with which we are confronted in applying a rule. In our approach, we take into account this in the framework of our various proof methods, that is, in the environment that allows us to reason forward, reason backward, reason in a mixture of them and so on. In Ref. 13), the metalanguage (ML) for interactive proof in LCF[12], a polymorphically typed, functional programming language, is used to show how logical calculi can be represented and manipulated within it. In Ref. 1), 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.

Our approach to a general reasoning system differs from the other ones cited above in three

respects. First, in EUODHILOS one can specify his or her own logic in a more direct and tractable way than others which require us to learn a formal system or meta-logic for encoding a logic. Second, much emphasis has been placed on reasoning facilities and proof methods which EUODHILOS should have in order to make proof construction more powerful and easier. Third, EHODHILOS has a unique reasoning-oriented interface not only for raising user-friendliness but also helping us conceive ideas for constructing the proofs. Dawson's generic logic environment[8] is very similar to our approach in many ways, but it only deals with logics in sequent presentations with all-introduction rules.

6. Concluding Remarks and Future Research Topics

In this paper, we have presented the unique features of a general-purpose reasoning assistant system EUODHILOS. We have shown the advantages and potential of our approach through a number of formal systems and their proof examples. Specifically, the following have been demonstrated:

(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. All the logics with their proofs were created in several hours. If we had had to develop a reasoning system with the same functions as EUODHILOS for each logic from scratch, it would have taken much time to do it, and we would have had to repeat almost the same task for constructing a reasoning system every time we were working on a new logic. EUODHILOS has demonstrated the usefulness of generality in a much wider range of applications[37].

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

The definite clause grammar formalism was employed for specifying logical syntax. We have found it more natural and easier for users to define a logical syntax, compared to the other approaches to logical system description languages mentioned before. And the DCG framework allowed us to automatically generate a parser with the function which generates the internal structure of an expression, and an unparser (generator). Therefore a user does not need to commit himself in those generations at all. Another positive feature is that the framework requires less expressive knowledge from the user in order to describe the logics. This shows the advantage of a logic programming approach to a general reasoning system. It is needless to say that the search and unification operations, which the logic programming have, are essential for traversing a search space for a proof and manipulating formulas and proofs, especially in a general setting for a general reasoning system.

The utilities such as a formula editor and a syntax checker to test an user-defined logical language are also provided to EUODHILOS and have been served to check the intended syntax. We have shown that the definite clause grammar formalism greatly lighten a user's burden in setting up his own language, together with those utilities[26].

(iii) Proving methodology based on sheets of thought

Lots of experiments for proving have convinced us that reasoning by several sheets of thought naturally coincides with human thought processes, such as analysis and synthesis in scientific exploration, from the part to the whole and vice versa. It may be also expected that they turn out to give a promising way towards proving in the large (see Ref. 38) for the detailed discussion about the proof methods of EUODHILOS).

(iv) Visual interface for reasoning

We have tried to analyze intrinsically how reasoning-oriented human-computer interface should be. However, it is not so easy to objectively assess the interface. We just have found that the visual interface for reasoning not only has been useful but also has served to easily define the logics and to conceive ideas for constructing the proofs.

An attempt at constructing a general-purpose reasoning assistant system is, however, at the initial stage of research and development, and lacks 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 improve EUODHILOS.

(a) Flexible proof architecture and proof representations

EUODHILOS forces us to handle various logic presentations in a single tree form of proofs. However, other proof trees as in Fitch style presentation of an axiomatic system seem not to be tractable within our proof architec-

ture. For example, modus ponens rule in Fitch style is represented as follows:

| formula1
|---
| formula2

formula1⊃formula2

This says that "formula1⊃formula2" is derivable if "formula2" is derived under the assumption "formula1". Another example is proof net, a natural deduction-like proof representation in linear logic. For these, A graphical drawing method to design proof architecture would be desirable.

Our tree form representation of proofs tends to make proof construction expand too much in both direction: horizontally and vertically. This tendency becomes crucial for large proof development. A better solution to proof representation might be to use tree form proofs in combination with a certain proof abridgement method and/or indented proof format.

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

Issues of designing a language for proof tactics/tacticals and amalgamating an object theory and a meta theory are inevitable, in particuler for the large proof development in applications. They would be helpful to attain the naturalness and efficiency of proofs at the same time.

It is also a remarkable recognition that reasoning generally consists of the manipulation of information, not symbols and they are just one of the many forms in which information can be couched[3),33)]. We believe that when we intrinsically consider reasoning it becomes crucial to incorporate such an aspect into syntactical reasoning.

(c) Theory revision and theory inheritance

Various theories or logics are involved in a larger 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 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. On the other hand, theory inheritance among theories is expected to yield a way to build up a large theory from its components since it could allow the theorems and proofs of a smaller and weaker theory to be inherited as those of a bigger and stronger theory. In doing so, we might need such a concept as theory morphism.

(d) 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. Especially our system will provide a settting for a general-purpose computer-aided learning system which is new and promising for learning various logics and solving reasoning tasks. Besides we are particularly interested in clarifying the feasibility of using EUODHILOS as a tool of logical model construction and a specialized use of EUODHILOS such as a reasoning tool for computer-aided programming.

Acknowledgements The first author would like to thank Prof. J. A. Robinson (Syracuse University), Prof. R. K. Meyer, Prof. M. A. McRobbie and Dr. J. K. Slaney (Australian National University) for their valuable comments and discussions on an earlier version of this paper.

Earlier versions of the paper were presented at the Automated Reasoning Project in Australian National University, Department of Computer Science in Victoria University of Wellington, Department of Computer Science in University of Queensland and Computer Laboratory in Cambridge University. This paper benefited from discussions at all these places.

This work is part of a major research and development of the Fifth Generation Computer System project conducted under a program set up by the MITI.

References

1) Abrial, J. A.: The Mathematical Construction of a Program, *Science of Computer Programming*, Vol. 4, pp. 45-86 (1984).

2) 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).

3) Barwise, J. and Etchemendy, J.: A Situation-Theoretic Account of Reasoning with Hyperproof (extended abstract), STASS Meeting (1988).

4) Batog, T.: *The Axiomatic Method in Phonology*, Routledge & Kegan Paul LTD (1967).

5) de Bruijn, N. G.: A Survey of the Project

automath, Seldin and Hindley (eds.), *To H. B. Curry : Essays on Combinatory Logic, Lambda calculus and Formalism*, pp. 579-606, Academic Press (1980).

6) Burkholder, L.: The Halting Problem, *SIGACT NEWS*, Vol. 18, No. 3, pp. 48-60 (1987).

7) Constable, R. L., et al.: *Implementing Mathematics with the Nuprl Proof Development System*, Prentice-Hall (1986).

8) Dawson, M.: A Generic Logic Environment, Ph. D. thesis, Dept. of Computing, Imperial College (1991).

9) Felty, A. and Miller, D.: Specifying Theorem Provers in a Higher-Order Logic Programming Language, *LNCS*, Vol. 310, pp. 61-80 (1988).

10) Fujimura, T.: Why Does Logic Matter to Philosophy?, *Philosophy of Science, Vol. 14, The Journal of Philosophy of Science Society, Japan*, pp. 1-5 (1981) (in Japanese).

11) Gallin, D.: *Intensional and Higher-Order Modal Logic, with Applications to Montague Semantics*, North-Holland (1975).

12) Gordon, M. J., Milner, A. J. and Wadsworth, C. P.: Edinburgh LCF, *LNCS*, Vol. 78, Springer (1979).

13) Gordon, M. J. C.: Representing a Logic in the LCF Metalanguage, Neel, D. (ed.), *Tools and Notions for Program Construction*, pp. 163-185, Cambridge U. P. (1982).

14) Griffin, T. G.: An Environment for Formal System, ECS-LFCS-87-34, Univ. of Edinburgh (1987).

15) Harper, R., Honsell, F. and Plotkin, G.: A Framework for Defining Logics, *Proc. of Symposium on Logic in Computer Science*, pp. 194-204 (1987).

16) Harel, D.: Dynamic Logic, Gabbay, D. and Guenthner, F. (eds.), *Handbook of Philosophical Logic, Volume II : Extensions of Classical Logic*, pp. 497-604, D. Reidel (1984).

17) Hoare, C. A. R.: An Axiomatic Basis for Computer Programming, *CACM*, Vol. 12, No. 10, pp. 576-580, 583 (1969).

18) Ketonen, J. and Weening, J. S.: EKL—An Interactive Proof Checker, User's Reference Manual, Dept. of Computer Science, Stanford Univ. (1984).

19) Kunst, J.: Making Sense in Music I—The Use of Mathematical Logic, *Interface*, Vol. 5, pp. 3-68 (1976).

20) Langer, S. K.: A Set of Postulates for the Logical Structure of Music, *Monist*, Vol. 39, pp. 561-570 (1925).

21) Martin-Löf, P.: *Intuitionistic Type Theory*, Bibliopoplis (1984).

22) Matsumoto, Y., Tanaka, H., Hirakawa, H., Miyoshi, H. and Yasukawa, H.: BUP: A Bottom-up Parser Embedded in Prolog, *New Generation Computing*, Vol. 1, pp. 145-158 (1983).

23) Meyer, R. K.: A General Gentzen System for Implicational Calculi, *Relevance Logic Newsletter*, Vol. 1, No. 3, pp. 189-201 (1976).

24) 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).

25) Minami, T., Sawamura, H., Satoh, K. and Tsuchiya, K.: EUODHILOS: A General-Purpose Reasoning Assistant System—Concept and Implementation—, *LNCS 383*, pp. 172-187, Springer-Verlag (1990).

26) Ohashi, K., Yokota, K., Minami, T., Sawamura, H. and Ohtani, T.: An Automatic Generation of a Parser and an Unparser in the Definite Clause Grammar, *Trans. IPS Japan*, Vol. 31, No. 11, pp. 1616-1626 (1990). (in Japanese)

27) Parker, J. H.: Social Logics: Their Nature and Uses in Social Research, *Cybernetica*, Vol. 25, No. 4, pp. 287-307 (1982).

28) Paulson, L. C.: The Foundation of a Generic Theorem Prover, *J. of Automated Reasoning*, Vol. 5, pp. 363-397 (1989).

29) Peirce, C. S.: *Collected Papers of C. S. Peirce*, Hartshorne, Ch. and Weiss, P. (eds.), Harvard Univ. Press (1974).

30) 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, *Artif. Intell.*, Vol. 13, pp. 231-278 (1980).

31) Prawitz, D.: *Natural Deduction*, Almqvist & Wiksell (1965).

32) Reps, T. and Alpern, B.: Interactive Proof Checking, *ACM Symp. on Principles of Programming Languages*, pp. 36-45 (1984).

33) Robinson, J. A.: Private communication (1989).

34) Sawamura, H.: A Proof Constructor for Intensional Logic, with S5 Decision Procedure, IIAS R. R., No. 65 (1986).

35) Sawamura, H. and Minami, T.: Conception of General-Purpose Reasoning Assistant System and Its Realization Method, 87-SF-22, WGFS, IPS, 1987 (in Japanese)

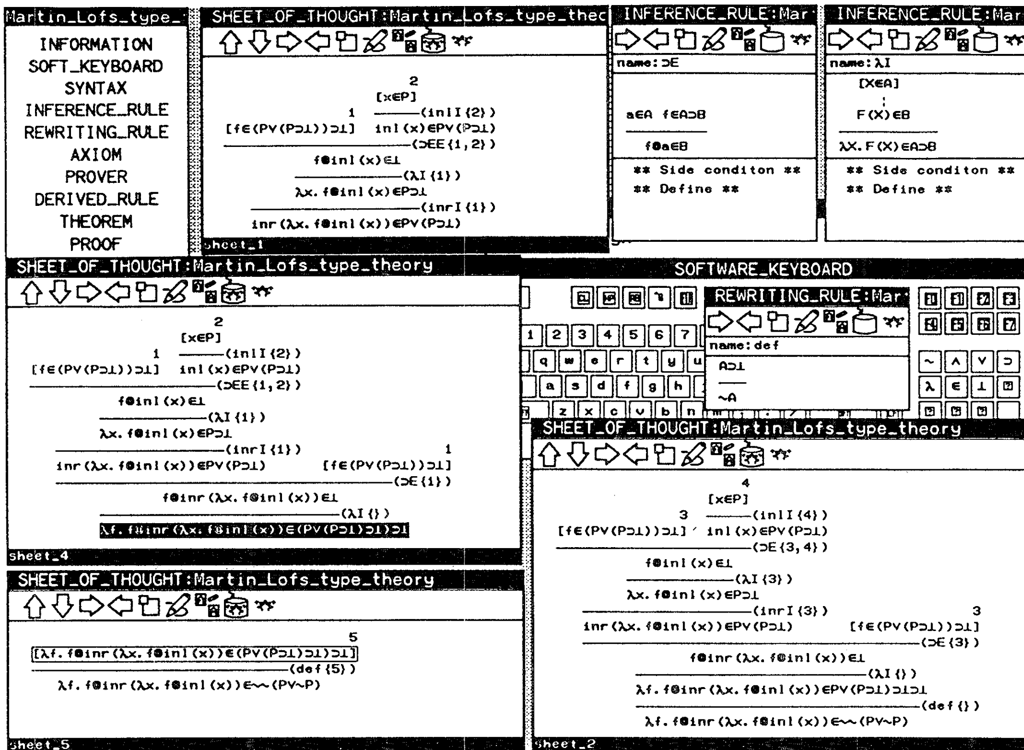
36) Sawamura, H.: Specifying General Logics and Constructing Proofs: A Case Study in EUODHILOS (1992). (in preparation)

37) Sawamura, H., Minami, T., Ohtani, T., Yokota, K. and Ohashi, K.: A Collection of Logical Systems and Proofs Implemented in EUODHILOS I, IIAS-RR-91-13E, Fujitsu Lab. (1991).

38) Sawamura, H., Minami. T. and Ohashi, K.: Proof Methods based on Sheet of Thought in EUODHILOS, IIAS-RR-92, Fujitsu Lab. (1992).
39) Sawamura, H., Minami, T. and Meyer, R. K.: Representing a Logic in EUODHILOS, IIAS-RR-92, Fujitsu Lab. (1992). (in preparation)
40) Slaney, J.: A General Logic, *Australasian J. of Philosophy*, Vol. 68, No. 1, pp. 74-88 (1990).
41) Thistlewaite, P. B., McRobbie, M. A. and Meyer, R. K.: *Automated Theorem-Proving in Non-classical Logics*, Pitman Publishing (1988).
42) Turner, A.: *Logics for Artificial Intelligence*, Ellis Horwood Limited (1984).
43) Trybulec, A. and Blair, H.: Computer Assisted Reasoning with MIZAR, *IJCAI'85*, pp. 26-28 (1985).
44) Weyhrauch, R. W.: Prolegomena to a Theory of Mechanized Formal Reasoning, *Artif. Intell.*, Vol. 13, pp. 133-179 (1980).
45) Zanardo, A. and Rizzotti, M.: Axiomatization of Genetics 2. Formal Development, *J. Theoretical Biology*, Vol. 118, pp. 145-152 (1986).

(Received August 31, 1992)
(Accepted October 13, 1994)

Appendix 1. Intuitionistic Type Theory and a Constructive Proof



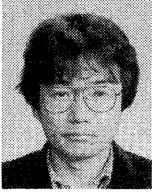
The EUODHILOS system consists of two major parts: one for defining a user's logical system and the other for constructing proofs on sheets of thought. Most of the interaction is performed using a mouse, though some facilities such as syntax/rule editors clearly require keyboard input.

The screen only displays some sheets of thought which appeared in the example proof process of the theorem, the double negation of the excluded middle. Each sheet is a special window surmounted by a title and a row of command buttons (icons) pointed at by means of a mouse. Four icons from the left allow the user to scroll up, down, left and right respectively. The fifth icon allows the user to resize a sheet of thought. The sixth button actually has four modes to which there appear four

icons ; pencil and eraser icons for proof editing, and up and down arrows for indicating the direction of reasoning : forward and backward. The seventh icon represents copy and move modes. The two icons from the right allow the user to save and quit, or quit without save respectively.

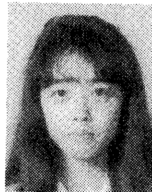
On the screen are laid four sheets of thought at work, in addition to the logic menu, two inference rules, a rewriting rule and software keyboard. The sheet_1 contains a proof fragment. It is expanded in a few proof steps as seen in the sheet_4. The sheet_5 contains a proof fragment which was obtained by applying a rewriting rule "def" to our goal backward. The final proof on sheet_2 is obtained by connecting the conclusion highlighted on sheet_4 to the premise encompassed by a rectangular frame on sheet_5.

Appendix 2. Hoare Logic and a Partial Correctness Proof of a Program



Hajime Sawamura received the B. E., M. E. and Doctor of Engineering degrees from Hokkaido University in 1972, 1975 and 1993 respectively. Since 1980 he has been with Institute for Social Information Science, Fujitsu Laboratories Ltd., where he is currently a research fellow of computational logic group. During 1990-1991, he was a visiting fellow of Australian National University. His research interests include computational logic, logical foundation of computer software and artificial intelligence. He is a member of IPSJ, JSAI, JSSST, and Philosophy of Science Society of Japan.

Toshiro Minami received the B. E. degree in Electronics from Kyushu Institute of Technology in 1973 and the M.S. in Mathmatics from Kyushu University in 1975. He has been working for Institute for Social Information Science, Fujitsu Laboratories Ltd. since 1984. He was a visiting fellow of Australian National University in 1993. His research interests include computational logic, category theory and artificial intelligence. He is a member of IPSJ and EATCS.



Kaoru Yokota received the B. A. degree in psychology from Keio University in 1985. From 1985 to 1992, she worked at Fujitsu Laboratories Ltd. She was engaged in the research and development of reasoning assistant system.

Kyoko Ohashi received the Bachelor degree in mathematics from Tsuda College in 1986. In 1986, she joined Fujitsu Laboratories Ltd. She was engaged in the research and development of reasoning assistant system. Her current research interests are object-oriented analysis and repository. She is a member of IPSJ.