ICOT Technical Report: TR-0748

TR-0748

Proof Methods based on Sheet of Thought in EUODHILOS

by
H. Sawamura, T. Minami& K. Ohashi

March, 1992

© 1992, ICOT

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome

(03)3456-3191～5
Telex ICOT J32964
Minato-ku Tokyo 108 Japan

Institute for New Generation Computer Technology

Proof Methods based on Sheet of Thought in EUODHILOS*

Hajime Sawamura, Toshiro Minami

International Institute for Advanced Study of Social Information Science (IIAS),
FUJITSU LABORATORIES LTD., 140 Miyamoto, Numazu, Shizuoka 410-03, JAPAN
{hajime, minami}@iias.flab.fujitsu.co.jp

and

Kyoko Ohashi

Software Laboratory, FUJITSU LABORATORIES LTD.,
1015 Kamikodanaka, Nakahara-ku, Kawasaki, Kanagawa 211, JAPAN
kyoko@soft.flab.fujitsu.co.jp

Abstract

In recent years there has been a growing interest in using computers as an aid for manipulating formal systems. EUODHILOS is a general reasoning system for a variety of logics which has an expressive framework enough to represent a wide variety of logics, aiming at a tractable formalism for a wide class of users. In this paper, we propose proof construction facilities and flexible proof methods on sheet of thought (proving methodology in a broader sense) which is one of the unique features of EUODHILOS. Sheet of thought is an environment for reasoning in such a way that proofs are interactively constructed in a tree-form through reasoning forward or backward, connecting or separating several proof fragments and so on. Taking up the intuitionistic type theory, we illustrate a proof process led by some of the proof methods and proving facilities.

---

* Part of this work was done while the first author was with the Automated Reasoning Project of Australian National University.

1. Introduction

Mathematical methods of the kind studied in logic are extensively used and applied to a broad class of questions of a logical nature(e.g., [Turner 84], [Genesereth 87]). Therefore much research has been devoted to the theory and methodology of computer-assisted reasoning, and to building computer systems for proving theorems automatically, for checking proofs and for developing proofs interactively in specific logical systems(e.g., [Gordon 79], [Ketonen 84], [Constable 86]).

In recent years, however, there has been a growing interest in using computers as an aid for manipulating formal systems(e.g., [Harper 87], [Griffin 87], [Felty 88], [Paulson 89]), as well as in formalizing general formal systems which yields a unified account of a fairly wide range of logical systems(e.g., [Meyer 76], [Belnap 82], [Avron 87], [Meseguer 87], [Feferman 89], [Slaney 90], [Gabbay 90]).

Our system EUODHILOS* [Sawamura 87, 88, 90, 91] is a logic-independent proof editor and constructor. More specifically, it is a general-purpose reasoning assistant system that allows users interactively to define the language and derivation rules of a logical system relevant for the universe of discourse under consideration, and to construct proofs in the defined system. 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 emphases have 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 [Dawson 91] is very similar to our approach in many ways, but it only deals with logics in sequent presentations with all-introduction rules.

So far in the area of reasoning by a computer, there has been little actual work in proof methods(proving methodology in a broader sense) for reasoning systems. This is in contrast to

---

* This acronym abbreviates the phrase, Every Universe Of Discourse Has Its LOgical Structure[Langer 25].

the current situation in software engineering. In this paper, as a first step towards proving methodology or proof engineering, we propose proof construction facilities and flexible proof methods on sheet of thought which allow us to reason in a natural and efficient way. Sheet of thought is one of the unique features of EUODHILOS and is an environment for reasoning in such a way that proofs are interactively constructed in a tree-form through reasoning forward or backward, connecting or separating several proof fragments and so on.

The remainder of the paper is organized as follows. In Section 2, we first describe an outline of EUODHILOS. In Section 3, various proof construction facilities and proof methods are described. In Section 4, taking up the intuitionistic type theory[Martin-Löf 84], we illustrate a proof process led by some of the proof construction facilities and proof methods described in Section 3. The final section contains an evaluation of our reasoning methods, obtained by various proof experiments with other logical systems[Sawamura 91].

2. Outline of functional features of EUODHILOS

First of all, we describe an outline of functional features of EUODHILOS. We think it would be helpful for readers to understand an overall image of EUODHILOS and the positioning of issues to be treated in this paper. EUODHILOS supports problem solving in logical methods like this: Suppose we have a problem of a logical nature and want to solve it with a logic. Then what we have to do next is to describe the problem and reason about it in our own logic. However the task of implementing any particular logic on a machine is quite demanding and logics usually vary with problem domains. In such a situation, EUODHILOS helps us to specify our logic and to construct a proof in it. In fact, EUODHILOS is a general-purpose reasoning assistant system that allows users interactively to define the syntax and inference rules of a formal system and to construct proofs in the defined system. In developing such a general reasoning system, we incorporate three important components:

(1) an expressive and tractable framework to represent a logic,

(2) a powerful and flexible proof construction facility, and

(3) a visual reasoning-oriented human-computer interface.

The framework for representing a logic is needed to specify the user's logic which consists of a language system for symbols, terms, formulas, etc. and a derivation system for axioms, inference rules, etc. Well-known definite clause grammar formalism[Pereira 80] is slightly augmented and employed for specifying the syntax of logical expressions. Once the syntax is defined, EUODHILOS generates a parser and an unparser for the language, as well as the internal structures of expressions in the defined syntax[Ohashi 90]. Together with the syntax debugger and the structure editor for formulas, this greatly lightens a user's burden in setting up his or her own language.

The derivation system consists of axioms, inference rules, and rewriting rules. An inference rule is specified in natural deduction style 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. 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. 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. 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. A rewriting rule is specified with a pair of forms before and after rewriting. EUODHILOS automatically generates many possible forms of an expression which may be obtained by successive applications of a given rewriting rule. Many well-known styles of logical stipulation can be treated within this framework - for example, Hilbert's, Gentzen's, equational, and even tableau styles.

The major drawback of reasoning in formal logic is that derivations tend to be lengthy and tedious because of the level of detail required in reasoning. In addition, performing formal derivations is time consuming and error prone. Using computers for formal reasoning should help to overcome these problems. EUODHILOS has various facilities which support the natural and efficient constructions of proofs in a defined logical system. Foremost among these facilities is the idea of a "sheet of thought"; moreover, proofs are in tree form, and various proof methods are used. A sheet of thought is a field in which we can compose a proof from its fragments, separate a proof into parts, or reason using derived rules and lemmas which have already been

established. EUODHILOS supports various proof methods related to sheets of thought - forward reasoning, backward reasoning, mixed forward and backward reasoning, and schematic proof. The proofs are visualized in tree-form with justifications indicated in the right margin. An interface with automated theorem provers is also provided for EUODHILOS in order to augment its reasoning facilities.

To make the system user-friendly and easy to use, we have put a lot of effort into designing the user interface. EUODHILOS includes a formula editor, font editor, software keyboard, electronic stationery for reasoning, and other features for ease of use.

Exploiting the bit-map display with multi-window environment, mouse, icon, pop-up-menu, etc., EUODHILOS is implemented in ESP (an object-oriented Prolog) on PSI-II(III)/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 object-logic is to be constructed as an instance object of a logic class.

3. Proof construction facilities and proof methods in EUODHILOS

In the area of software engineering, much work has been devoted to the issues such as how programs should be developed formally or informally, and programming environment which promotes reliable program development. However, less effort has been paid to proof development methods. In the subsection 3.1, we present proof construction facilities which may be expected to lead to such important issues as proof modularization, abstract or schematic proofs, and visualization of proof structures. These have some resemblance to those which appear as principles of programming. In the subsection 3.2, various proof styles suitable for human reasoners are presented . More generally, they are expected to constitute reasoning (proving) methodology, which reminds us of programming methodology.

3.1 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. 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 expected 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 [Peirce 74]. 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*

In EUODHILOS, inference and rewriting rules are presented in a natural deduction style as follows :

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

together with the speculation of side conditions, 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. 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 A $\wedge$ B is obtained by the rule named $\wedge$I from the two premises A and B, and it depends on the assumptions named 1 and 2 above the premises.

$$\frac{\begin{array}{cc}\vdots & \vdots \\ A & B\end{array}}{A \wedge B}\ (\wedge I\ \{1, 2\})$$

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

(3) *Schema (meta) variables*

The Schema variables are introduced to EUODHILOS not only for the schematic specifications of axioms, inference rules or rewriting rules, but also for derived rules or schematic proofs in general. 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 operate on schema variables, in addition to the usual variables.

3.2 Proof methods

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 reasoning so far has been 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.

(1) *Input of logical expressions*

Derivations begin with inputting 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.

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

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

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

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[Gordon 79, 88] 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 [Tarski 56].

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

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 mentioned above.

Besides, connection methods such as analogical matching, instantiation, etc., 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.

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

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

$$\dfrac{[A]^1 \quad [A \supset B]^2}{B}(\supset I\ [1, 2\}) \text{ and } \dfrac{\dfrac{[B]^4}{C}(\ldots\ \{4\}) \quad [D]^5}{E}(\ldots\ \{4, 5\})$$

(6) Automated reasoning

In principle, there can be no mechanized way of provability 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 (see Figure 2 in Section 4 below in which the button "PROVERS" interfaces with outside facilities). 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.

4. An illustrative example in intuitionistic type theory

In this section, along the line of computer session is shown a concrete proof process using some of those proof construction facilities and proof methods discussed in the previous section.

EUODHILOS starts with the following logic/theory menu window (Figure 1) which displays the entries to creating a new logic, designing new symbols, logic calculator, ending the system, and various logics already defined.

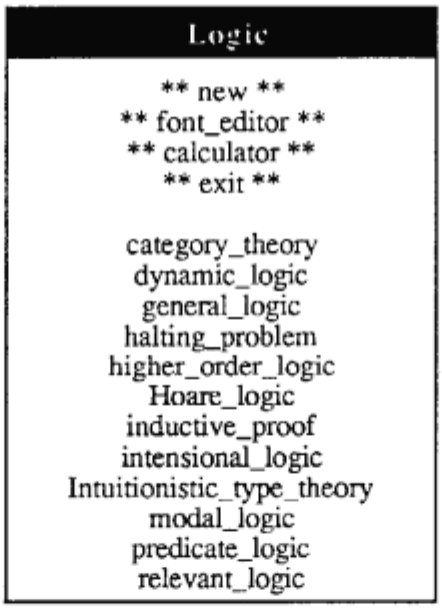


Figure 1. Logic/theory menu at start-up

Selecting an entry on the logic/theory menu brings forth a presentation of the corresponding logic. EUODHILOS views a logic as consisting of the following basic items and environment: logic information, software keyboard, syntax, inference rules, rewriting rules, axioms, derived rules, provers, theorem database, proof sheet called sheet of thought, as can be seen in Figure 2.

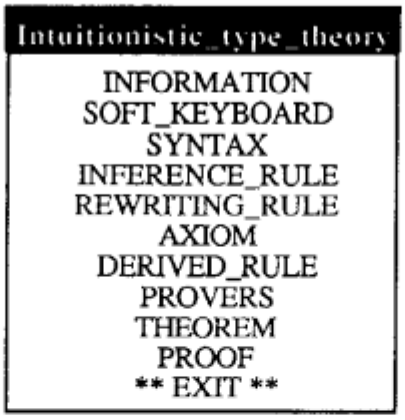


Figure 2. Basic items of a logic

In this section, we take up a subset of intuitionistic type theory [Martin-Löf 84][Backhouse 88] and exemplify a proof construction process of a constructive proof in it.

4.1 Specifying a logic

We begin with specifying the language system to be used in intuitionistic type theory(see [Sawamura 92] for the details of representing a logic in EUODHILOS). If we need special symbols for our language, we can easily design them using the font editor and assign them to the software keyboard. Then we proceed to specifying the syntax of logical expressions in terms of definite clause grammar(DCG) formalism. This is slightly augmented with special constructs to handle logical concepts such as variable binding, scope and so on[Ohashi 90]. The full syntax definition of a subset of intuitionistic type theory is given in Figure 3. The language definition basically consists of three parts: an object language, a meta language and an operator definition.

SYNTAX : Intuitionistic_type_theory

save make test structure print reshape exit

```
% Meta_language
meta_term --> meta_term1;
meta_type -> "A"|"B";
meta_term1 --> "F"|"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.
```

Figure 3. Syntax of intuitionistic type theory

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 and the predicate, e.g. "inl" in the term "inl(x)", are listed simply by themselves or the nonterminals by which they are denoted. Let's take a look at the principal expression called 'judgement' in intuitionistic type theory, which is defined to be a string generated by the following clause (rule);

judgement --> term, "∈", type ;

where the nonterminal "term" is an expression in λ-calculus and the nonterminal "type" is a proposition in the well-known "propositions-as-types interpretation". The judgement reads "the term is a proof of the type (the proposition)". It can be seen that the judgement is naturally and well described in the framework of definite clause grammar formalism (DCG) and that it's almost the same as the usual context-free grammar formalism (CFG). Once the logical syntax has been defined like this, then a (bottom-up) parser for it which constructs the internal structures of expressions is automatically generated as well as its unparser(see [Ohashi 90] for the details of the algorithms). The parser and unparser generated are internally used in all the succeeding phases of symbol manipulations. Here we can test whether the language just specified is what we intended or not, by using our syntax debugger and the structure editor for formulas[Ohashi 90].

An inference rule is specified in natural deduction style in three parts: the premises of the rule, the conclusion of the rule, and side conditions. The side conditions are the restrictions that may be imposed on the derivations of the conclusion from the premises and assumptions, such as variable occurrence conditions and substitution conditions. Then EUODHILOS can check those side conditions automatically in the proof process. In EUODHILOS, a deductive system is represented simply in such a descriptive apparatus that is usually used in a logic textbook without relying on any other descriptive framework, as can be seen in Figure 4 where four inference rules are specified. These are the rules for function introduction and elimination, and the two rules for ∨–introduction.

INFERENCE RULE: Intuitionistic_type _theory
name: λ–I
$\begin{array}{l}[X \in A] \\ \vdots \\ F(X) \in B \\ \hline \lambda X.F(X) \in A \supset B\end{array}$
** side condition**

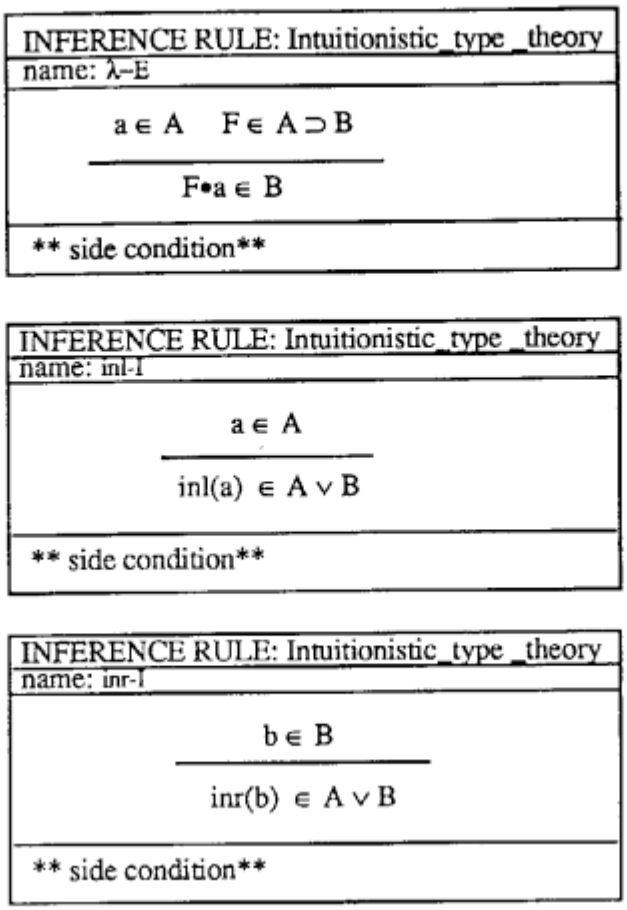


Figure 4. Inference rules of intuitionistic type theory

All of these rules have no side conditions with them. The λ–introduction rule means "if the premise F(X) ∈ B (meaning F(X) is a proof of B) is derived under the assumption x ∈ A (meaning x is a proof of A), then we can derive that λX.F(X) is a proof of A ⊃ B". Then the assumption "[X ∈ A]" encompassed by brackets is discharged from the assumptions on which the conclusion "λX.F(X) ∈ A ⊃ B" depends.

A rewriting rule is specified with a pair of forms before and after rewriting. Figure 5 specifies the definition "~A = A ⊃ ⊥" as a rewriting rule.

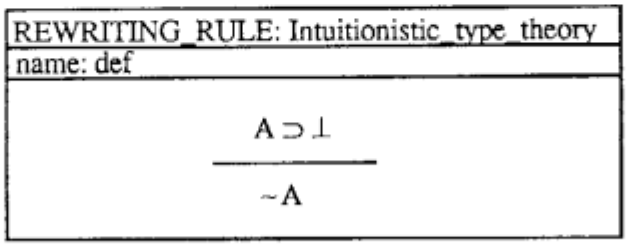


Figure 5. Rewriting rule

In EUODHILOS, a rewriting rule can be applied in both directions: downward and upward, according to the user's indication.

4.2 Constructing a proof in the specified logic

The logic (formal system) of intuitionistic type theory has been settled. In this section, we illustrate the construction of a simple proof in this logic under our flexible proof construction facilities and methods. A proof is interactively constructed in tree-form on proof sheets called sheets of thought, through reasoning forward or backward, connecting or separating several proof fragments and so on. Here we'll show how to construct the theorem "~~(p ∨ ~p)", which is the double negation of the law of excluded middle.

First of all, suppose we have a sheet of thought containing a partially constructed proof fragment, which would turn out to be a part of the final proof. As can be seen in Figure 6, the conclusion in this tree is obtained under the assumption 2 as the dependency list {2} shows.

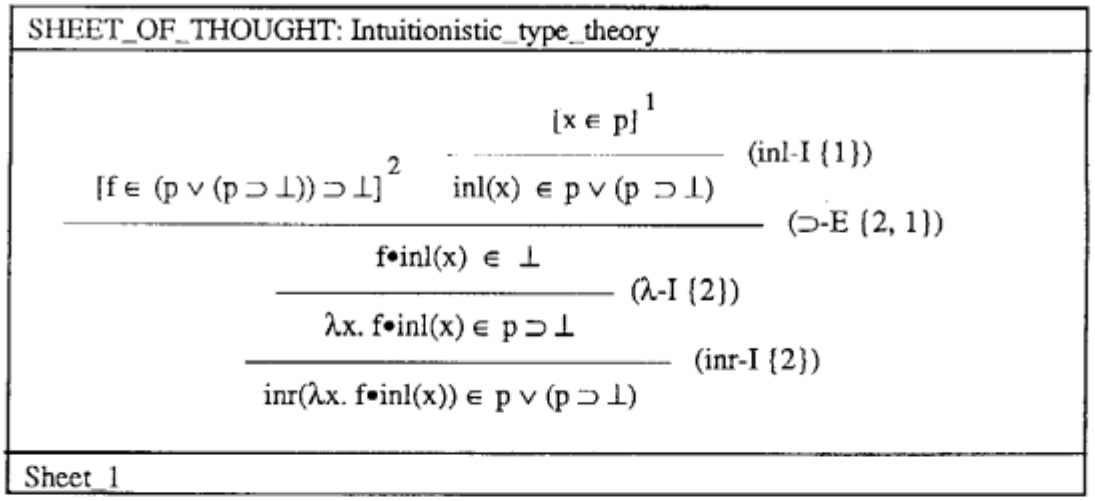


Figure 6. A proof fragment on sheet of thought

On this sheet, first let us copy the assumption "f ∈ (p ∨ (p ⊃ ⊥)) ⊃ ⊥ " numbered 2 into the right hand side of the proof tree. Copy can be done by selecting the expression to copy and clicking the place to be copied by mouse. The result is shown in Figure 7.

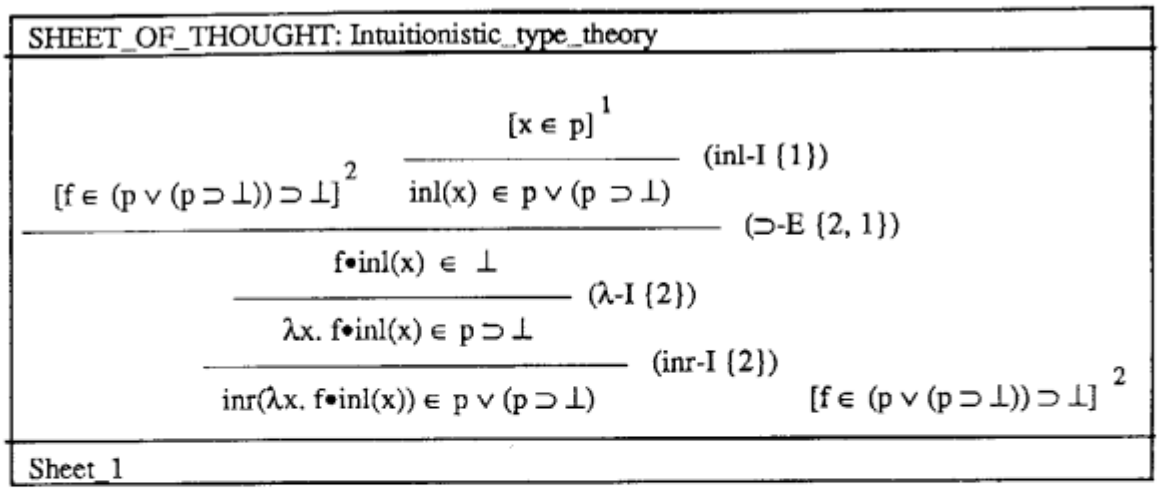


Figure 7. Copying an expression

Then apply the implication-elimination rule to the two lowest judgements. Application of a rule can be done by selecting judgements to be applied and choosing a rule to apply from the rule menu. As the result we get the judgement "f•inr(λx. f•inl(x)) ∈ ⊥", as shown in Figure 8.

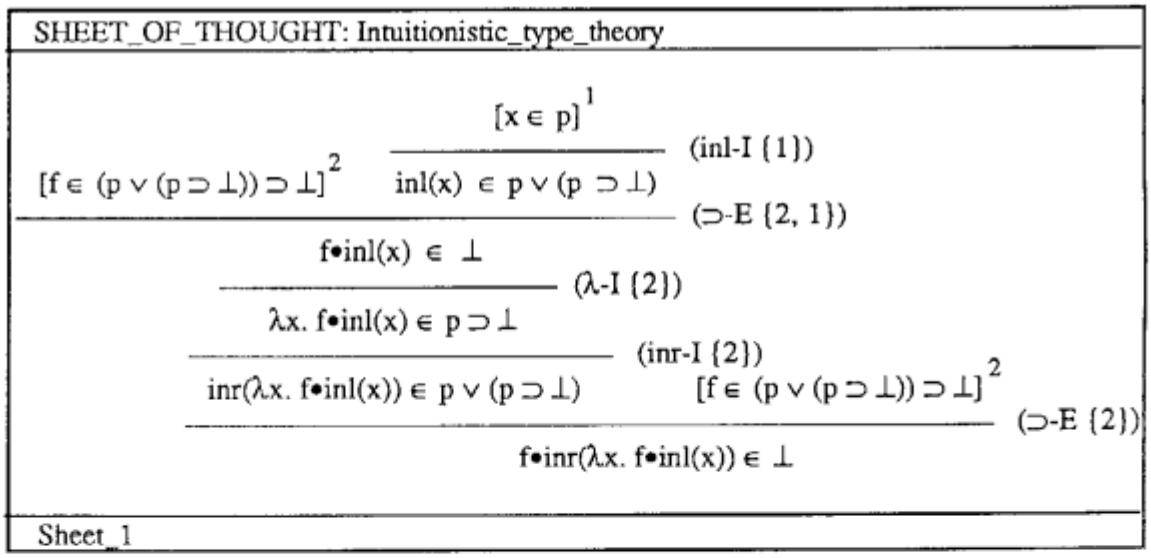


Figure 8. Application of a rule (⊃–E)

Next if we apply the λ-introduction rule to the lowest judgement under the assumptions numbered 2, then we obtain the judgement "λf. f•inr(λx. f•inl(x)) ∈ (p ∨ (p ⊃ ⊥) ⊃ ⊥) ⊃ ⊥", as shown in Figure 9.

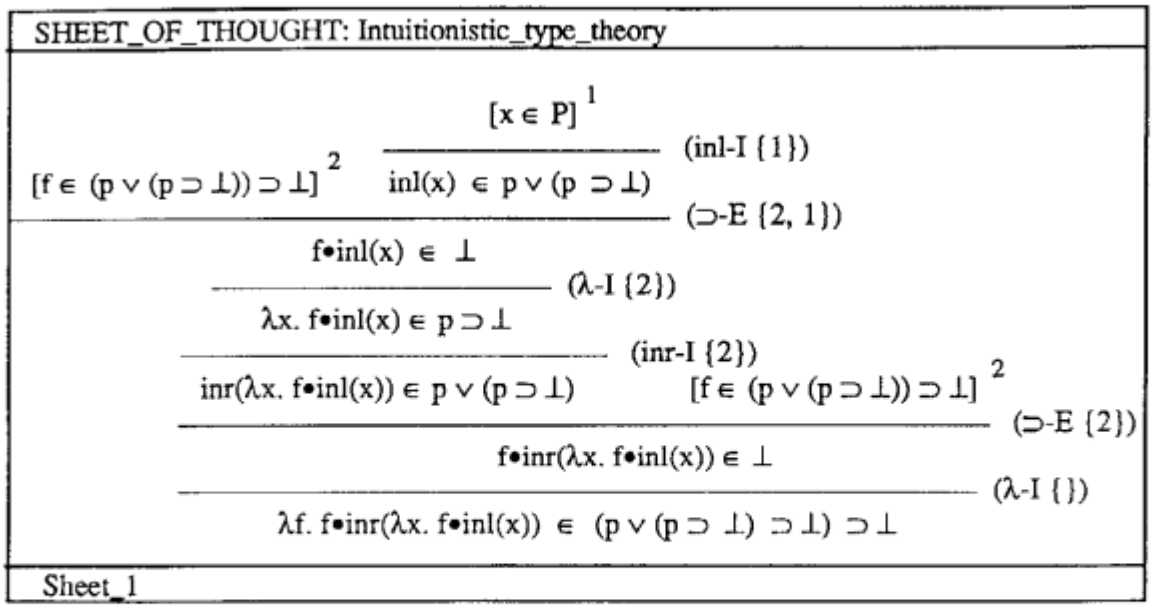


Figure 9. Application of a rule (λ-I)

Obviously the assumptions have been discharged as the empty justification { } shows.

We have almost arrived at the desired conclusion. But in order to make it simpler, first let us copy the conclusion on a new sheet (see in Figure 10) and then apply the rewriting rule def as a definition of "~" to it.

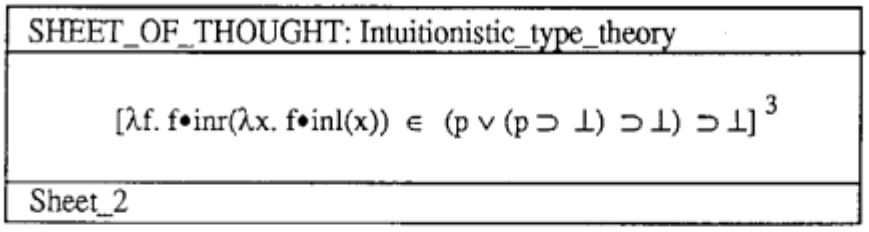


Figure 10. Copy of the lowest judgement on a new sheet of thought

In doing so, we set the rewriting counter to 3 because the right hand side of the expression seems to deserve rewriting up to three times. EUODHILOS generates possible forms of the expression obtained by successive applications of this rewriting rule. Thus we get "λf. f•inr(λx. f•inl(x)) ∈ ~~(p ∨ ~p)", as shown in Figure 11.

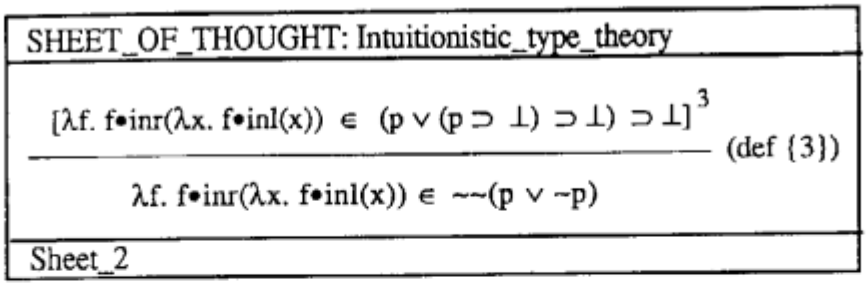


Figure 11. Result of a rewriting rule def

At this stage, we can connect the two proof fragments into one by indicating the conclusion in the sheet 1 and the assumption on the sheet 2 because they coincide with each other. To do so, first we choose the two judgements to be connected and then just double click the left mouse button.

Readers may wish to try the proof along with the following line. Suppose we have already had another sheet of thought containing a schematic goal like this, "F ∈ ~~(p ∨ ~p)", where F is a meta-variable which stands for an unknown term to be instantiated later. This judgement obviously can be expanded backward from bottom to top by using the rewriting rule named "def", as shown in Figure 12.

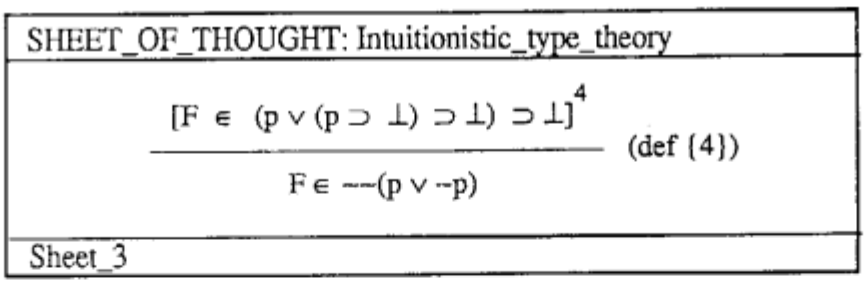


Figure 12. A schematic proof

As you may have noticed, the assumption is analogous to the result in the previous proof fragment, so let's instantiate to the meta-variable F the term "λf. f•inr(λx. f•inl(x))". Then we get the instantiated proof fragment like this;

SHEET_OF_THOUGHT: Intuitionistic_type_theory

$$\frac{[\lambda f.\ f\bullet inr(\lambda x.\ f\bullet inl(x)) \in (p \vee (p \supset \bot) \supset \bot) \supset \bot]^4}{\lambda f.\ f\bullet inr(\lambda x.\ f\bullet inl(x)) \in \sim\sim(p \vee \sim p)}\ (\text{def } \{4\})$$

Sheet_3

Figure 13. Instantiation of a proof tree in Figure 12

At this stage, we can connect these proof fragments into one as before.

In this manner, we have finally reached our desired proof. The sheet of thought in Figure 14 displays the whole proof tree for it and thus we've shown that the double negation of the excluded middle cannot be refuted in intuitionistic type theory.

SHEET_OF_THOUGHT: Intuitionistic_type_theory

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{[f \in (p \vee (p \supset \bot)) \supset \bot]^2 \qquad \cfrac{[x \in P]^1}{inl(x) \in p \vee (p \supset \bot)}\ (\text{inl-I } \{1\})}{f\bullet inl(x) \in \bot}\ (\supset\text{-E } \{2, 1\})}{\lambda x.\ f\bullet inl(x) \in p \supset \bot}\ (\lambda\text{-I } \{2\})}{inr(\lambda x.\ f\bullet inl(x)) \in p \vee (p \supset \bot)}\ (\text{inr-I } \{2\}) \qquad [f \in (p \vee (p \supset \bot)) \supset \bot]^2}{f\bullet inr(\lambda x.\ f\bullet inl(x)) \in \bot}\ (\supset\text{-E } \{2\})}{\lambda f.\ f\bullet inr(\lambda x.\ f\bullet inl(x)) \in (p \vee (p \supset \bot) \supset \bot) \supset \bot}\ (\lambda\text{-I } \{\})}{\lambda f.\ f\bullet inr(\lambda x.\ f\bullet inl(x)) \in \sim\sim(p \vee \sim p)}\ (\text{def } \{\})$$

Sheet_1

Figure 14. A proof tree of the law of double negation of the excluded middle

5. Concluding remarks

This paper revealed proof construction facilities and flexible proof methods on sheet of thought (proving methodology in a broader sense), one of the issues involved in the construction of an

interactive yet powerful general reasoning system for a variety of logics. It should be noted that one of the points in this paper is that our proof construction facilities and proof methods are not tailored to any specific logics, but designed in a generic way and hence can be applied to various logics as well.

We have applied EUODHILOS to various types of reasoning(see [Sawamura 91] for the details). Logic and proof examples include the following. (a) First-order logic: various pure logical tautologies, the unsolvability of the halting problem, inductive proof, hardware verification, and category theory. (b) Second-order logic: the equivalence between the principle of mathematical induction and the principle of complete induction. (c) Propositional modal logic: modal reasoning about programs. (d) Intensional logic: (1) the reflective proof of a metatheorem; (2) Montague semantics of natural language. (e) Martin-Löf's intuitionistic type theory(see Appendix 1): various constructive proofs. (f) Hoare logic and dynamic logic: reasoning about program properties. (g) General logic. (h) Relevant logics. (i) A logic of knowledge.

Although we would not say that the examples tried in the above proof experiments are large and complex enough to evaluate the proof methods of EUODHILOS (except a few proofs which became 100 ~ 1000 steps long in primitive inference steps), we have become convinced particularly of the following points:

(i) Proof visualization in tree-form

Visualizing proofs in tree-form made proof structures much clearer than the linear format of a proof did. It was found that in this representation of a proof tree there would be less chance of getting lost as to where we are now in the proof and what we should do next.

(ii) Proof modularization on several sheets of thought

Similar to program modularization, proof development by proof modularization allows us to construct a proof in a structured way. Sheets of thought have induced a modularized proof development to attain well managed and efficient proof construction. In fact, they support such a proof construction that the main part of a proof which contains a conclusion is located in a sheet of thought and every subgoal which may be successfully connected to the main part of a proof tree is located in each sheet of thought. It may be expected that they turn out to give a promising way towards proving in the large.

Lots of experiments for proving have also convinced us that reasoning by several sheets of thought goes well with human thought processes, such as analysis and synthesis in scientific exploration, from the part to the whole and vice versa.

(iii) Various proof styles

Goal-directed reasoning is usually a predominant style for computer reasoning. However, it is not always efficient for one to reason only in a backward way. In our experiences with EUODHILOS, we found forward reasoning and its mixture with backward reasoning more efficient and natural, in particular for large and complex proof development.

(iv) Ease of use

EUODHILOS was designed so as to be usable by people who may not be so familiar with computer systems. We can specify our own logic and develop proofs under an easy to use environment based on a graphical interface that provides a controlled dialogue between the user and EUODHILOS.

This paper have dealt with only syntactical aspects of reasoning by a computer, not a whole spectrum of reasoning tasks. The following issues should be attacked as important but most challenging aspects of reasoning which should be incorporated into syntactical reasoning systems like EUODHILOS: (a) 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[Barwise 88]. As a matter of fact, we usually make crucial use of diagrams, graphs or other non-linguistic form of representation, since we are adept at reasoning with and making inferences directly from them, (b) When one studies and appreciates the power of a proof, it is tactics and elegance and so on which seize one's attention and stay in one's memory. We should put more emphasis on studying the nature of human-oriented reasoning patterns and capabilities, and the epistemic structure of proofs regardless of the logic on which they are based[Robinson 90].

Acknowledgements

We would like to thank Prof. J. A. Robinson (Syracuse University), Dr. R. K. Meyer, Prof. M. A. McRobbie and Dr. J. K. Slaney (Australian National University) for their valuable comments and discussions on our work. The paper also benefited from the discussions with Dr. D. Basin (University of Edinburgh). Special thanks are owed to Mrs. K. Yokota for her collaboration on the development of EUODHILOS.

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

[Avron 87] Avron, A.: Simple Consequence Relations, ECS-LFCS-87-30(1987), Univ. of Edinburgh.

[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.

[Barwise 88] Barwise, J. and Etchemendy, J.: A situation-theoretic account of reasoning with Hyperproof (extended abstract), STASS Meeting, 1988.

[Belnap 82] Belnap, N. D, Jr.: Display Logic, J. of Philosophical Logic 11(1982), pp. 375-417.

[Constable 86] Constable, R.L., et al.: Implementing mathematics with the Nuprl proof development system, Prentice-Hall, 1986.

[Dawson 91] Dawson, M.: A generic logic environment, Ph.D. thesis, Dept. of Computing, Imperial College, 1991.

[Felty 88] Felty, A. and Miller, D.: Specifying theorem provers in a higher-order logic programming language, LNCS, Vol. 310, pp. 61-80, 1988.

[Feferman 89] Feferman, S.: Finitely inductively presented logics, in Ferro, R., Bonotto, C., Valentini, S. and Zanardo, A.(eds.): Logic Colloquium '88, N-Holland, pp. 191-220, 1989.

[Gabbay 90] Gabbay, D.: Labelled deductive systems, CIS-Bericht-90-22, University of München, 1990.

[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 88] Gordon, M. J.: HOL: A proof generating system for higher-order logic, in Birtwistle, G. and Subrahmanyam, P. A.(eds.): VLSI Specification, Verification and Synthesis, Kluwer Academic Publishers, pp. 73-128, 1988.

[Griffin 87] Griffin, T. G.: An environment for formal system, ECS-LFCS-87-34, Univ. of Edinburgh, 1987.

[Harper 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.

[Ketonen 84] Ketonen, J. and Weening, J. S.: EKL - An interactive proof checker, User's reference manual, Dept. of Computer Science, Stanford Univ., 1984.

[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, Bibliopoplis, 1984.

[Meseguer 87] Meseguer, J.: General Logics, in H. D. Ebbinghaus et al.(editors): Logic Colloquium '87, North-Holland(1987), pp. 275-329.

[Meyer 76] Meyer, R. K.: A General Gentzen System for Implicational Calculi, Relevance Logic Newsletter, Vol. 1, No. 3(1976), pp. 189-201.

[Ohashi 90] 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, Transactions of Information Processing Society of Japan, Vol. 31 , No. 11, pp. 1616-1626, 1990 (in Japanese).

[Paulson 89] Paulson, L. C.: The foundation of a generic theorem prover, J. of Automated Reasoning, Vol. 5, pp. 363-397, 1989.

[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.

[Robinson 90] Robinson, J. A.: Natural and artificial reasoning, in Arbib, M. A. and Robinson, J. A.(eds.): Natural and Artificial Parallel Computation, The MIT Press, pp. 277-309, 1990.

[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).

[Sawamura 88] Sawamura, H., Minami, T., Yokota, K. and Ohashi, K.: Potential of a general-purpose reasoning assistant system EUODHILOS, in I. Nakata and M. Hagiya(eds.): Software Science and Engineering, Selected Papers from the Kyoto Symposia, World Scientific Pub., pp. 164-188, 1991.

[Sawamura 90] Sawamura, H., Minami, T., Yokota, K. and Ohashi, K.: A Logic Programming Approach to Specifying Logics and Constructing Proofs, Proc. of the Seventh International Conference on Logic Programming, edited by D. H. D. Warren and P. Szeredi, The MIT Press, pp. 405-424, 1990.

[Sawamura 91] 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.

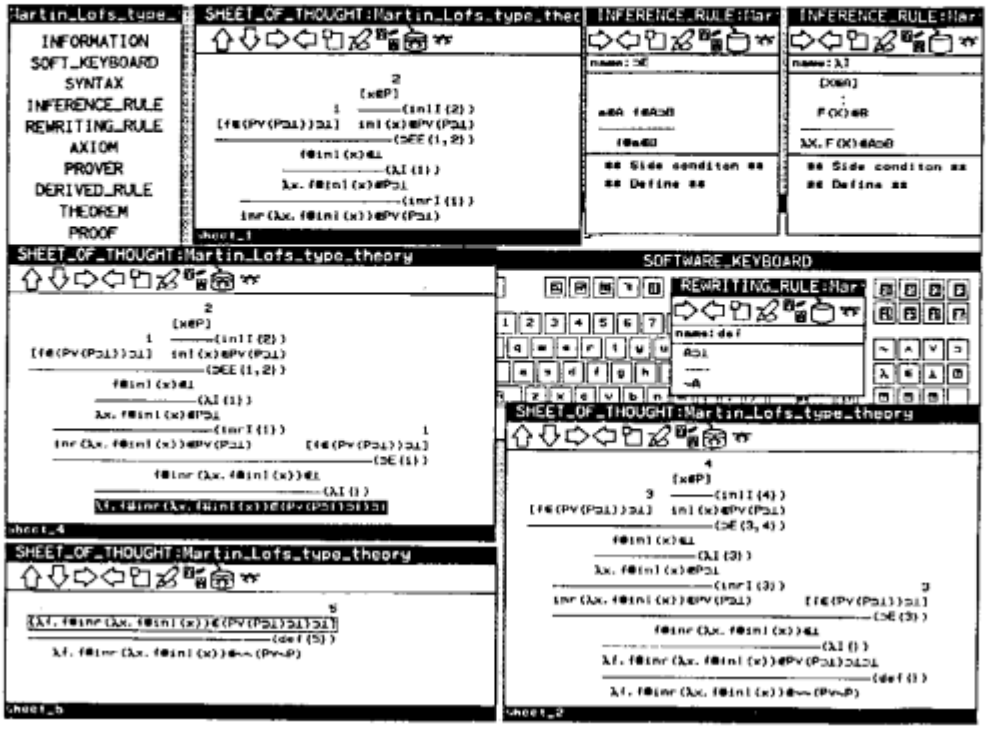
[Sawamura 92] Sawamura, H., Minami, T. and Meyer, R. K.: Representing a Logic in EUODHILOS, IIAS-RR-92, Fujitsu Lab., 1992.

[Slaney 90] Slaney, J.: A General Logic, Australasian J. of Philosophy, Vol. 68, No. 1, pp. 74-88, 1990.

[Tarski 56] Tarski, A.: On the concept of logical consequence, in Logic Semantics Metamathematics, Oxford Univ. Press, 1956.

[Turner 84] Turner, A.: Logics for artificial intelligence, Ellis Horwood Limited, 1984.

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. The screen only displays some sheets of thought which appeared in the example proof process.

Each sheet is a special window surmounted by a title and a row of command buttons(icons) pointed by 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 allows the user to save and quit, and 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, sheet-4, sheet_5 and sheet_2 correspond to Fig. 6, Fig. 9, Fig. 11 and Fig. 14 in the body of the paper respectively.

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.