TR-582

A Logic Programming Approach to Specifying Logics and Constructing Proofs

by
H. Sawamura, T. Minami, K. Yokota
& K. Ohashi (Fujitsu)

August, 1990

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191 ~5
Telex ICOT J32964

A Logic Programming Approach to Specifying Logics and Constructing Proofs

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

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

* Present address : Automated Reasoning Project, Research School of Social Sciences, Australian National University, GPO Box 4, Canberra, ACT 2601, AUSTRALIA
hs@arp.anu.oz

Kaoru Yokota **Kyoko Ohashi**
FUJITSU LABORATORIES LTD.
1015 Kamikodanaka, Nakahara-ku, Kawasaki, Kanagawa 211, JAPAN

Abstract

Logic programming languages serve as good implementation languages 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. This paper describes a practical logic programming approach to a general-purpose reasoning assistant system that allows a user to define his/her own logical system and to reason in the defined system.

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

1. Introduction

A new dimension of computer-assisted reasoning research is explored in this paper, employing a logic programming approach. Logic programming languages serve as good implementation languages 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 [9]. This paper describes a practical logic programming approach to a

general-purpose reasoning assistant system that allows a user to define his/her own logical system and to 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., [37], [36]), and surprisingly in aesthetics which has been thought of as being in a directly opposite position to logic (e.g., [20], [19]), as well as in other scientific theories (e.g., [4], [26], [41]). 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 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 [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 of these logics as possible and allows us to reason in various ways [34][24].

There are three major subjects to be pursued for such an interactive and general reasoning support system. One is a language expressive enough to describe a large class of logics. The second is the kind of reasoning styles suitable for human reasoners which should be taken into account. More generally, reasoning (proving) methodology, which reminds us of programming methodology, needs to be investigated. The third subject is reasoning-oriented human-computer interface that may be well established as an aspect of reasoning supporting facilities. An easy to use system with good interface would be helpful 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 under development is described. We emphasize the following three points : (1) a formal system description language based on definite clause grammar, (2) a methodology for proving based on several sheets for logical thought, (3) visual human-computer interface for reasoning. 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. They include first-order logic for a logical puzzle, an inductive proof and the halting problem, second-order logic, propositional modal logic, intuitionistic type theory, Hoare logic and dynamic logic for program verification, and the intensional logic for Montague's semantics .

2. Need, significance and design philosophy

Much work has been devoted to special-purpose reasoning assistant systems whose underlying logics are fixed (e.g., [12], [39], [18], [38], [7]). 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 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 [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 :

- Formal system description language based on the definite clause grammar (DCG)
- 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

3.1 Functional features

We list the main features of EUODHILOS and explain them briefly (see [40] for the details). We start by describing the language of a logic to EUODHILOS. Fundamentally, EUODHILOS has almost no defaults, so it must be told everything,

3.1.1 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 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 [33], λProlog in [9] and [23], typed λ-calculus with dependent types in [14] and [15], a specification language for a wide variety of logics in [1], an attribute grammar formalism in [31], a metalanguage ML in [13] and a higher-order logic in [27].

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 fomation rules. Also, notational conventions for defining or abbreviating symbols are usually 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". These can also be dealt with in a recursive fashion.

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

3.1.2 Specifying a logical syntax and the expressiveness of the definite clause grammar

In EUODHILOS, an object language system 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) [29], 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 [29]: (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 [31]. The necessity for context-dependency is often encountered in defining logical syntax. The following examples show how naturally and economically DCG allows us to express the context-dependency which occurrs in ordinary logical practice and allows arbitrary

tree structure to be built in the course of the parsing, with the help of (ii) and (iii).

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)) --> functor (f), "(", term(T), ",", term(S), ")", {arity(f, 2)}.

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).

It should be noted that DCG originally possesses the apparatus for describing the correspondence between the external expressions manipulated by a user and the internal expressions manipulated by a computer in terms of the parametrized nonterminal.

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 expressions. 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 [25]. The unparser 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., [14], [31]), 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 own language and taking care of it. Readers interested in the details of the algorithms can find these in [25].

3.1.3 Specifying a derivation system

A derivation system consists of an inference system and a rewriting system. They are given in a natural deduction style presentation [30] by a user. An inference rule, especially, is stated as a triple consisting of three elements, where the first is the derivations of the premises of a rule, the second the conclusion of a rule, and finally the third the restrictions that are imposed on the derivations of the premises and conclusion, such as variable occurrence condition (eigenvariable) and substitutability such as "t is free for x in P". Well-known typical styles of logic presentations such as Hilbert's style, Gentzen's style or the equational style can be treated within this framework.

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

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

where brakets 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. Then, the dependency of a conclusion on temporary assumptions is automatically calculated by the ordinary method [18]. In backward reasoning, discharging the asumptions, generating some assumptions and checking the application conditions are in general impossible and hence delayed until completing the partial proof tree under construction.

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. In a sense, they play a role of tactics, although we have not had operations for combining tactics to form tacticals [12].

Rewriting rules are useful for handling equational reasoning often appearing in ordinary mathematical practice. These can be simply presented to the system in the following schematic format :

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

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.

3.1.4 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 [28]. 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. Obviously, proving by the use of sheets of thought yields proof

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

(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. 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. Substitution and unification viewed as the common and primitive symbol operations are supposed to operate on schema variables, in addition to the usual variables.

3.1.5 Proving methodology

The predominant style of interactive reasoning 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.

As examples of deduction process on several sheets of thought, let us illustrate some of the reasoning styles in more detail.

(1) Forward and backward reasoning

In order to deduce forward by applying an inference rule, one has to start by selecting the formulas used as premises of the rule. Then one may select an appropriate inference rule from the rule menu which has been automatically generated at the time of logic definition, or one may input a formula as the conclusion. If one selects a rule, then the system applies the rule to the premises 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. 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 justificiation of those partially justified proof fragments is delayed until the completion of a final proof tree.

(2) Schematic reasoning

EUODHILOS allows us to construct an abstract proof in the sense that metavariables 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 a proving facility is very convenient for having an indeterminate or unknown predicate (such as invariant assertion in Hoare logic) unspecified temporarily in the proof constructing process.

(3) Reasoning by lemmas and derived rules

Theorems constructed on the sheet and validated derived rules can be stored in the theorem database and rule base respectively. They are referred to and reused in the later proofs for other theorems. After using EUODHILOS systematically and over a long period of time, the theorems turn out to build up theories.

(4) 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 can be viewed as valid :

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

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 the distributed proof fragments through an appropriate rule of inference. Let us take an example schema of modus ponens :

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

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 beome 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. So it is omitted.

3.1.6 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.

(1) Formula editor

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

(2) Software keyboard and Font editor

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

(3) Stationery for reasoning

Independently of 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.

3.2 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.

4. Experiments and experiences with EUODHILOS

We have tried to apply EUODHILOS to various types of reasoning. Logics and proof examples that we have dealt with so far on EUODHILOS include various pure logical formulas, the unsolvability of the halting problem and an inductive proof with first-order logic (NK), the equivalence between the principle of mathematical induction and the principle of complete induction with second-order logic, modal reasoning about programs with propositional modal logic (T), the reflective proof of a metatheorem and Montague's semantics of natural language with intensional Logic (IL), Martin-Löf's intuitionistic type theory, reasoning about program properties with Hoare logic and dynamic logic. 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 [35] for the detailed definition of each logic and proof examples in the experiments.)

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 [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 one 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 three parts: an object language, a meta language and an operator definition.

Syntax of object langauge :

```
judgement ---> term, "∈", type ;
term --> bind_op, variable, ".", term1 ;
bind_op --> "λ" ;
term --> term1 ;
term1 --> term1, "•", term2 ;
term1 --> term2 ;
term2 --> "(", term, ")" ;
term2 --> or_intro, "(", term, ")" ;
or_intro --> "inr" | "inl" ;
term2 --> variable ;
variable --> x;
type --> type, "⊃", type1 ;
type --> type1 ;
type1 --> type1, "∨", type2 ;
type1 --> type2 ;
type2 --> "~", type2 ;
type2 --> "(", type, ")" ;
type2 --> basic-type ;
basic-type --> "P" | "⊥" ;
```

Syntax of meta language :

```
term1 --> meta_term1, "(", term, ")" ;
term1 --> meta_term1;
term2 --> meta_term1;
type2 --> meta_type;
variable --> meta_variable;
meta_term1 --> "F" | "a" | "b";
meta_term1 --> meta_variable;
meta_type --> "A" | "B";
meta_variable --> "X" | "f".
```

Operator definition with or without precedence :

```
with_precedence
   ~; •; ∨; ⊃; λ; ∈;
without_precedence
   or_intro, meta_term1.
```

It is noted that the syntax definition for the meta language is provided for defining inference rules schematically, and the operators displayed under the heading "with_precedence" have precedence in the indicated order and the operators without precedence, e.g. "inl" in the term "inl(x)", are listed simply by themselves or the nonterminals by which they are denoted. 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 [21][2]. 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 $\vee$-introduction, and the rewrite rule in lieu of the definition $\sim A \equiv A \supset \bot$.

$$\frac{\begin{array}{c}[X \in A] \\ \vdots \\ F(X) \in B\end{array}}{\lambda X.\, F(X) \in A \supset B} \quad (\lambda\text{-introduction } (\lambda\text{-I}))$$

$$\frac{a \in A \qquad f \in A \supset B}{f \bullet a \in B} \quad (\supset\text{-elimination } (\supset\text{-E}))$$

$$\frac{a \in A}{inl(a) \in A \vee B} \quad (\text{inl-introduction } (\text{inl-I}))$$

$$\frac{b \in B}{inr(b) \in A \vee B} \quad (\text{inr-introduction } (\text{inr-I}))$$

$$\frac{A \supset \bot}{\sim A} \quad (\text{definition as rewriting rule})$$

We have specified both the language system and derivation system which are possibly sufficient to the proof in Appendix 1. 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. However, a revision 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.

In Appendix 1, the screen layout of the proof of the theorem

~~(P ∨ ~P) is shown, which means that the law of 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 ~~P is always constructively valid.

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 executing 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 framework of DCG, as well as the inference rules of Hoare logic.

In Appendix 2, we show the screen layout of the proof of the following partial correctness assertion of a factorial program,

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!".

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. The proof examples includes the following properties of a factorial program :

Termination :

$x \geq 0 \supset \langle z := 1 ; ((x > 0)? ; z := x \times z ; x := x - 1)^* ; (x = 0)? \rangle true$

Partial Correctness :

$x = n \supset [z := 1 ; ((x > 0)? ; z := x \times z ; x := x - 1)^* ; (x = 0)?](z = n!)$

Total Correctness :

$x \geq 0 \wedge x = n \supset \langle z := 1; ((x > 0)? ; z := x \times z ; x := x - 1)^* ; (x = 0)? \rangle (z = n!)$

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 is ingeniously proved with the help of the reflection principle ([39]).

$\vdash P:t \Rightarrow \vdash \forall x:a.\ P:t$ *(Generalization rule)*

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

$(\lambda p:(s,(e,t)).\exists x:e.(fish:(e,t) \bullet x:e \wedge p:(s,(e,t))\{x:e\})) \bullet$

$\hat{}\lambda y:e.(believe:((s,t),(e,t)) \bullet \hat{}(walk:(e,t) \bullet y:e) \bullet j:e)$

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 :

$\exists x{:}e.(fish{:}(e,t) \bullet x{:}e \wedge believe{:}((s,t),(e,t)) \bullet \hat{}(walk{:}(e,t) \bullet x{:}e) \bullet j{:}e).$

First-order logic (with NK)

(1) Smullyan's logical puzzles
(2) Unsolvability of the halting problem
(3) Proof by structural induction on list
(4) Category theory

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.)

Propositional modal logic (T) and modal reasoning about programs

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

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

5. Related work

Much work has been devoted to building the systems for checking and constructing formal proofs in various logical systems, e.g., see [5], [39], [18] and [38] for proof checker, see [12], [7] and [33] for proof constructor, see [6], [34], [14], [15] and [24] for general system of computer-aided reasoning. Here we will confine ourselves to various approaches to the general system for computer-assisted reasoning to which much attention have been recently paid. And also, we will have to restrict ourselves to seeing only the distinction of a formal system description language in each approach for the sake of the space limitation and 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 [33], Prolog is employed as a logic description language as well as an implementation language of a proof constructor. In [9] and [23], λ-Prolog, which is a higher-order version of Prolog and hence more expressive than Prolog, is proposed to specify theorem provers. In [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 [31], 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 [27], 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 have to attain this in the framework of our proving methodology, that is, in the environment that allows us to reason forward, reason backward, reason in a mixture of them and so on. The uniform treatment for them, however, is left open. In a current version of EUODHILOS, the problem of automatically checking the application condition of a rule is supplanted by presenting the rule condition to be checked to a user when he or she chooses and applies a rule with the conditions and entrusting the confirmation to a user. Note that this is not our final solution to that. In [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 [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 rigorious approach to program construction : to define a theory and then to apply it.

In addition to such a purely theoretical interest as what a general theory of logics is, an important benefit of these treatments of formal systems is, although their approaches are different, that logic-independent tools for proof editors, proof chekers, and proof constructors can be constructed. As to logic-dependent tools, we think that it would be better to provide them by designing an appropriate metalanguage such as ML [12].

Among the present general reasoning assistant sytems, it seems fair to say that it is only EUODHILOS and [8] that attempted to integrate such a distinctive feature as proving methodology plus logic defining capability, emphasizing visual interface for reasoning.

6. Concluding remarks and future work

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 develope 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.

(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.

A formula editor and a debugging facility to test the defined language serve to check the intended syntax. We have shown that these greatly lighten a user's burden in setting up his own language [25].

(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 large.

(iv) Visual interface for reasoning

It is not so easy to objectively assess the interface. But I believe 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.

(1) Augmentation of formal system description language

Much effort has to be spent on making the logic description language more expressive. For example, in the current framework, rule descriptions for tableau method, Fitch style presentation of an axiomatic system, some formulation of relevance logic, etc., seem not to be expressible. Furthermore, automatic mechanism for checking the side conditions of rules is not satisfactory as remarked in the previous section. To overcome these deficiencies, we would need some more powerful rule description language and method.

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

Developing a language for proof strategies, incorporating metatheory, etc., are important subjects since these could attain increasing the naturalness and efficiency of proofs. 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][32]. We believe that when we intrinsically consider reasoning it become crucial to incorporate such an aspect into syntactical reasoning.

(3) Maintaining a relational dependency among various theories

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

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

Acknowledgements

We would like to thank Prof. J. A. Robinson (Syracuse University), Dr. R. K. Meyer and Dr. M. A. McRobbie (Australian National University) for their valuable comments and discussions on the paper. Thanks are also due to referees, whose comments were helpful.

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.

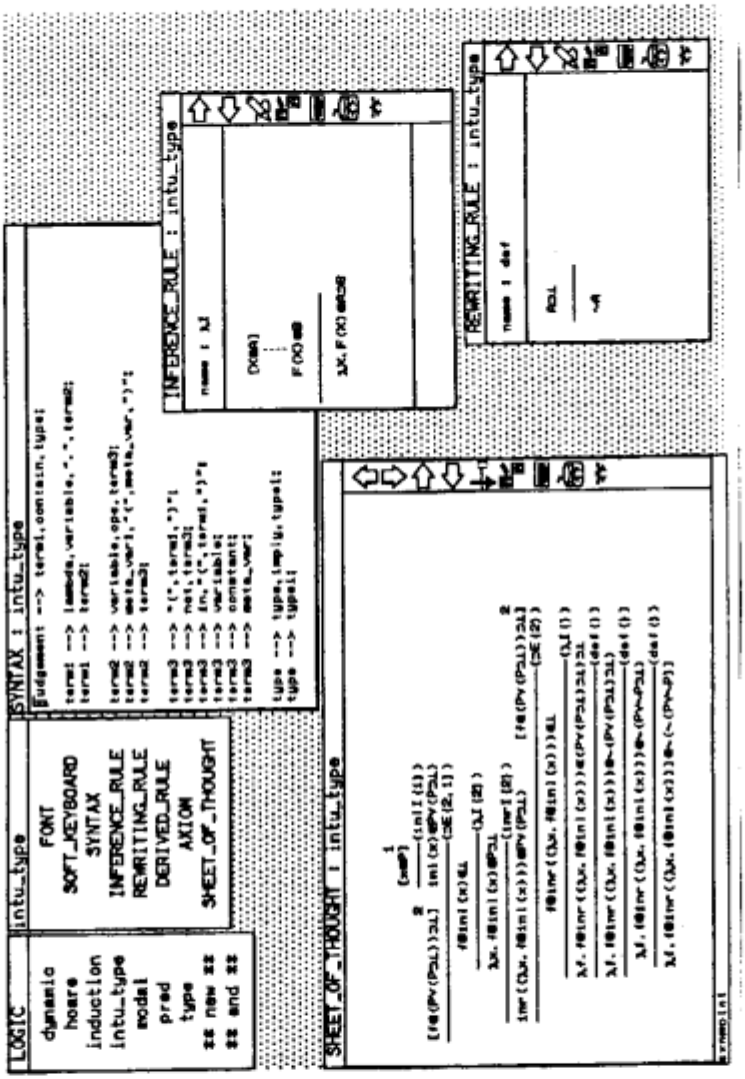
Dedicated with affection to the first author's late wife, Fumiko.

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, in: Seldin and Hindley (eds.), To H. B. Curry : Essays on Combinatoty logic, Lambda calculus and Formalism, Academic Press, pp. 579-606, 1980.
[6] Coquand, T and Huet, G. : Constructions : A higher order proof system for mechanizing mathematics, LNCS 203, pp. 151-184, 1985.
[7] Constable, R.L., et al. : Implementing mathematics with the Nuprl proof development system, Prentice-Hall, 1986.
[8] Dawson, M., Sadler, M. and Mainbaum, T.: Generic logic environment, Proc. of CASE '88, pp. 215-218, 1988.
[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, in: D. Neel (ed.), Tools and notions for program construction, Cambridge U. P., pp. 163-185, 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] Harrel, D. : Dynamic logic, in 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 baisis 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 5, pp. 3-68, 1976.

[20] Langer, S. K. : A set of postulates for the logical structure of music, Monist 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, Vo. 1, pp. 145-158, 1983.
[23] 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.
[24] Minami, T., Sawamura, H., Satoh, K. and Tsuchiya, K. : EUODHILOS : A general-purpose reasoning assistant system - concept and implemetation - , to appear in LNCS, Springer, 1988.
[25] Ohashi, K., Yokota, K., Minami, T. and Sawamura, H. : An automatic generation of a parser and an unparser in the definite clause grammar(submitted), 1989 (in Japanese).
[26] Parker, J. H. : Social logics : Their nature and uses in social research, Cybernetica, Vol. 25, No. 4, pp. 287-307, 1982.
[27] Paulson, L. C. : The foundation of a generic theorem prover, J. of Automated Reasoning, Vol. 5, pp. 363-397, 1989.
[28] Peirce, C. S. : Collected Papers of C. S. Peirce, Ch. Hartshorne and P. Weiss (eds.), Harvard Univ. Press, 1974.
[29] Pereira, F. C. N. and Warren, D. H. D. : Definite clause grammars for language analysis - A survey of the formalism and a comparison with augumented transition networks, Artificial Intelligence, Vol. 13, pp. 231-278, 1980.
[30] Prawitz, D. : Natural deduction, Almqvist & Wiksell, 1965.
[31] Reps, T and Alpern, B : Interactive proof checking, ACM Symp. on Principles of Programming Languages, pp. 36-45, 1984.
[32] Robinson, J. A. : Private communication, 1989.
[33] Sawamura, H. : A proof constructor for intensional logic, with S5 decision procedure, IIAS R. R., No. 65, 1986.
[34] Sawamura, H. and Minami, T. : Conception of general-purpose reasoning assistant system and its realization method, 87-SF-22, WGFS, IPS, 1987. (In Japanese).
[35] Sawamura, H., Minami, T, Yokota, K. and Ohashi, K. : Potential of a general-purpose reasoning assistant system EUODHILOS, Proc. of Software Science and Engineering, RIMS, Kyoto University, 1988, also IIAS Research Report, 1990.
[36] Thistlewaite, P. B., McRobbie, M. A. and Meyer, R. K. : Automated theorem-proving in non-classical logics, Pitman Publishing, 1988.
[37] Turner, A. : Logics for artificial intelligence, Ellis Horwood Limited, 1984.
[38] Trybulec, A. and Blair, H. : Computer assisted reasoning with MIZAR, IJCAI '85,pp. 26-28, 1985.
[39] Weyhrauch, R. W. : Prolegomena to a theory of mechanized formal reasoning, Artificial Intelligence, Vol. 13, pp. 133-179, 1980.
[40] Yokota, K., Ohashi, K., Sawamura, H. and Minami, T. : General-purpose reasoning assistant system EUODHILOS - its unique functions and implementation -, 1990 (in preparation) (in Japanese).
[41] Zanardo, A. and Rizzotti, M. : Axiomatization of genetics 2. Formal development, J. Theoretical Biology, Vol. 118, pp. 145-152, 1986.

Appendix 1. Intuitionistic type theory and a constructive proof



Appendix 2. Hoare logic and correctness proof of a program