ICOT Technical Report: TR-642

TR-642

A Collection of Logical System and Proofs Implemented in EUODHILOS I

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

May, 1991

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

Institute for New Generation Computer Technology

RESEARCH REPORT IIAS-RR-91-?E

A Collection of Logical Systems and Proofs Implemented in EUODHILOS I

Hajime Sawamura, Toshiro Minami, Takeshi Ohtani
Kaoru Yokota and Kyoko Ohashi

International Institute for Advanced Study of Social Information Science,
FUJITSU LABORATORIES LTD.

Numazu office
140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan
Telephone: (Numazu) 0559-23-2222 Telex: 3922508J Fax: 0559-24-6180

Tokyo office
17-25, Shinkamata 1-chome, Ohta-ku, Tokyo 144, Japan
Telephone: (Tokyo) 03-3735-1111 Telex: J28448FUJITSU Fax: 03-3730-6830

A Collection of Logical Systems and Proofs Implemented in EUODHILOS I *

Hajime Sawamura, Toshiro Minami, Takeshi Ohtani

International Institute for Advanced Study of Social Information Science,
FUJITSU LABORATORIES LTD.
140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan
E-mail: {hajime,minami,ohtani}@iias.flab.fujitsu.co.jp@uunet.uu.net

Kaoru Yokota and Kyoko Ohashi

Software Lab., FUJITSU LABORATORIES LTD.
1015 Kamikodanaka, Nakahara-ku, Kawasaki, Kanagawa 211,Japan
E-mail: {kaoru,kyoko}@sdl.flab.fujitsu.co.jp@uunet.uu.net

Abstract

A number of logical systems and theories have been implemented using a general-purpose reasoning assistant system EUODHILOS. These systems range from classical logic to non-classical logic and have been used or devised in computer science, artificial intelligence and other related fields. This document records how each of these systems has been treated. It helps readers to understand both the potential and actual usefulness of EUODHILOS in a much wider range of applications. It also serves as a guide to specifying logics and constructing proofs using EUODHILOS.

Key words: Logic, First order logic, Second order logic, Constructive type theory, Hoare logic, Dynamic logic, Intensional logic, Modal logic, General logic, Category theory, Halting problem, Induction, Tag calculus

*Part of this work had been done during the first author's stay at the Automated Reasoning Project, Australian National University.

Contents

1 Introduction

EUODHILOS is a general-purpose reasoning assistant system that allows users to interactively define the language and derivation rules of a logical system relevant for a universe of discourse under consideration, and to construct proofs in the defined system [Sawamura 90, Minami 90, Sawamura 91a]. This acronym is constructed by using the first letters of the phrase, Every Universe Of Discourse Has Its LOgical Structure [Langer 25]. Nowadays, the methods of mathematical logic provide a paradigm in computer science and artificial intelligence. Mathematical methods of the kind studied in logic are extensively used and applied to a broad class of questions of a logical nature.

To develop such a general-purpose reasoning assistant system, we decided there are three important components: an expressive logical system description language, a powerful and flexible proof construction facility, and a visual reasoning-oriented human-computer interface.

The logical system description language 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 is employed for specifying the syntax of logical expressions. Once defined, a parser and an unparser for the language are automatically generated as well as the internal structures of expressions in the defined syntax. This greatly lightens a user's burden in setting up one's own language [Ohashi 90]. The derivation system consists of axioms, inference rules, and rewriting rules. An inference rule is specified in a natural deduction style [Prawitz 65] with three items: 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 condition and substitutability. A rewriting rule is specified with a pair of forms before and after rewriting. Well-known styles of logic presentation such as Hilbert's, Gentzen's, or Equational could be treated within this framework.

The major drawback of reasoning in formal logic is that derivations tend to be lengthy and tedious because of the detailed level required in reasoning. In addition, performing formal derivations is time consuming and error prone. Using computers for formal reasoning is expected to overcome these problems. EUODHILOS has various facilities which support the natural and efficient constructions of proofs in a defined logical system, that is, sheet of thought, proving methodology and tree-form proof. A sheet of thought is a field of thought where we are allowed to compose a proof from its fragments, to separate a proof, or to reason using lemmas. EUODHILOS supports various proving methods based on several sheets of thought, that is, forward reasoning, backward reasoning, reasoning with their mixture, and schematic proof. The proofs are visualized in a tree-form with justifications indicated in the right margin.

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, software keyboard, stationery for reasoning, and other features for ease of use.

A number of logical systems and theories have been implemented using EUODHILOS.

These systems range from classical logic to non-classical logic and have been used or devised in computer science, artificial intelligence and other related fields. This document records how each of these systems has been treated. The purpose of this paper, then, is twofold:

(1) helping readers to understand both the potential and actual usefulness of EUODHILOS in a much wider range of applications, and

(2) serving as a guide to specifying logic and constructing proofs using EUODHILOS as well as a supplement of a users manual of EUODHILOS [Minami 91].

In what follows, we list a language system, derivation system and some proof examples with screen layout, for each logical system.

2 First-order logic (NK)

The first logic we take up is, of course, first-order logic. The axiom system is a well-known natural deduction NK [Prawitz 65] which was formalized by Gentzen in 1935.

2.1 Language system of first-order logic

Object language

```
formula --> formula,"≡", formula1;
formula --> formula1;
formula1 --> formula1, "⊃", formula2;
formula1 --> formula2;
formula2 --> formula2, "∨", formula3;
formula2 --> formula3;
formula3 --> formula3, "∧", formula4;
formula3 --> formula4;
formula4 --> "(", formula, ")";
formula4 --> "∼", formula4;
formula4 --> bind_op, variable, formula4;
formula4 --> basic_formula;

basic_formula --> "⊥"
basic_formula --> predicate_symbol1,"(",term, ")" ;
basic_formula --> predicate_symbol2,"(",term, ",",term,")" ;
basic_formula --> predicate_symbol3,"(",term, ",",term,",",term,")" ;

term --> constant;
term --> variable;

predicate_symbol1 --> "A"|"B"|"C";
predicate_symbol2 --> "H"|"O";
predicate_symbol3 --> "D"|"H";
constant --> "g"|"b"|"a"|"c"|"m"|"e";
variable --> "x"|"y"|"z"|"w"|"u"|"v";
bind_op --> "∀"|"∃";
```

Meta language

```
meta_formula --> "P"|"Q"|"R"|"S"|"T";
meta_term --> "S"|"T"|"X"|"Y"|"Z"|meta_term,"/",meta_term;
```

Interface between object and meta languages

```
term --> meta_term;
basic_formula --> meta_formula;
constant --> meta_term;
variable --> meta_term;
predicate_symbol1 --> meta_formula.
```

Operator definition

```
operator
    "/"; ("~"; bind_op); "∧"; "∨"; "⊃"; "≡";
predicate
    predicate_symbol1, predicate_symbol2, predicate_symbol3.
```

2.2 Derivation system of first-order logic

Inference rules (NK)

$$[\sim \mathrm{I}]\ \dfrac{\begin{array}{c}[A]\\ \vdots\\ \bot\end{array}}{\sim A} \qquad [\sim \mathrm{E}]\ \dfrac{A \quad \sim A}{\bot} \qquad [\sim\sim \mathrm{E}]\ \dfrac{\sim\sim A}{A}$$

$$[\wedge \mathrm{I}]\ \dfrac{A \quad B}{A \wedge B} \qquad [\wedge \mathrm{E1}]\ \dfrac{A \wedge B}{A} \qquad [\wedge \mathrm{E2}]\ \dfrac{A \wedge B}{B}$$

$$[\vee \mathrm{I1}]\ \dfrac{A}{A \vee B} \qquad [\vee \mathrm{I2}]\ \dfrac{B}{A \vee B} \qquad [\vee \mathrm{E}]\ \dfrac{A \vee B \quad \begin{array}{c}[A]\\ \vdots\\ C\end{array} \quad \begin{array}{c}[B]\\ \vdots\\ C\end{array}}{C}$$

$$[\supset \mathrm{I}]\ \dfrac{\begin{array}{c}[A]\\ \vdots\\ B\end{array}}{A \supset B} \qquad [\supset \mathrm{E}]\ \dfrac{A \quad A \supset B}{B}$$

$$[\forall \mathrm{I}]\ \dfrac{A(a)}{\forall x A(x)}\ a\text{:eigenvariable} \qquad [\forall \mathrm{E}]\ \dfrac{\forall x A(x)}{A(t)}$$

$$[\exists \mathrm{I}]\ \dfrac{A(t)}{\exists x A(x)} \qquad [\exists \mathrm{E}]\ \dfrac{\exists x A(x) \quad \begin{array}{c}[A(a)]\\ \vdots\\ C\end{array}}{C}\ a\text{:eigenvariable}$$

2.3 Unsolvability proof of the halting problem

Here we attempt to prove the unsolvability of the halting problem which can be conducted entirely within the resources of standard first-order predicate logic specified above [Burkholder 87]. The following symbolization is used in stating the problem in the natural deduction setting:

$A(x)$: x is an algorithm.
$C(x)$: x is a computer program in some programming language.
$D(x, y, z)$: x is able to decide whether y halts given input z.
$H(x, y)$: x halts given input y.

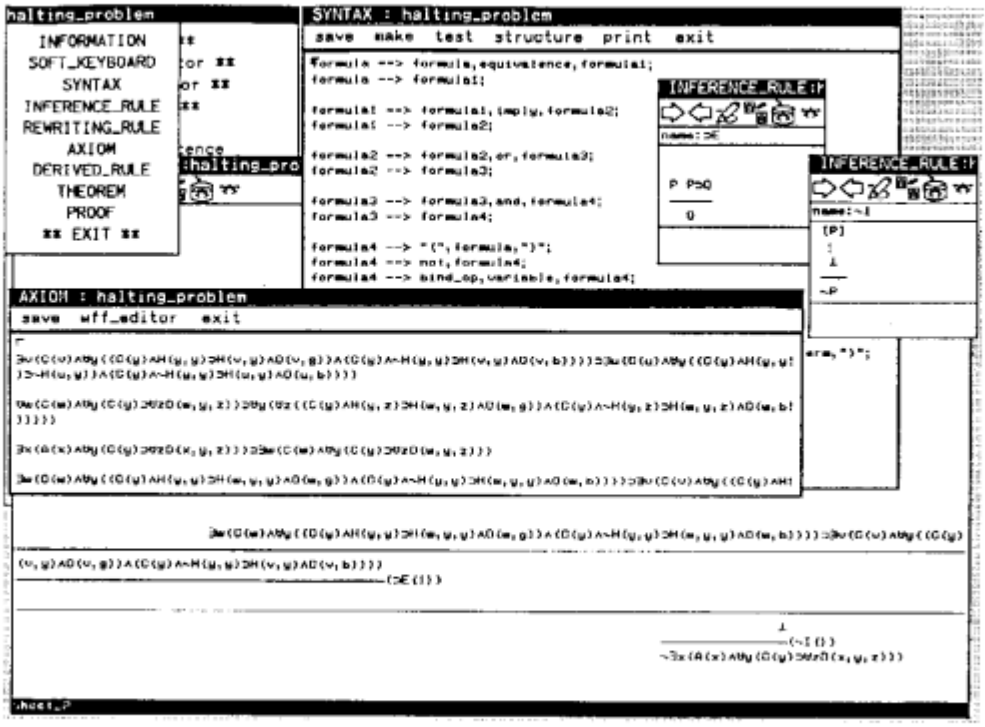


Figure 1: An unsolvability proof of the halting problem

$H(x,y,z)$: x halts given as inputs y and z.
$O(x,g)$: x outputs "g" (meaning "it halts").
$O(x,b)$: x outputs "b" (meaning "it does not halt").

Then Figure 1 displays the proof of the following theorem:

$$\vdash \sim\exists x(A(x)\&\forall y(C(y) \supset \forall z D(x,y,z)))$$

(no algorithm to solve the halting problem exists)

under the premises:

1. $\exists x(A(x)\&\forall y(C(y) \supset \forall z D(x,y,z))) \supset \exists w(C(w)\&\forall y(C(y) \supset \forall z D(w,y,z)))$
(Church's thesis)

2. $\forall w(C(w)\&\forall y(C(y) \supset \forall z D(w,y,z)) \supset \forall y\forall z((C(y)\&H(y,z) \supset H(w,y,z)\&O(w,g))$
$\&(C(y)\& \sim H(y,z) \supset H(w,y,z)\&O(w,b))))$

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

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

3 Constructive Type Theory

This system is due to Martin-Löf [Martin-Löf 84]. Originally each type is defined by the following four rules.

1. Formation Rule
   This rule says that we can form a certain type from certain other types or families of types.
2. Introduction Rule
   This rule says what are the canonical elements of the type.
3. Elimination Rule
   This rule shows how we may define functions on the type defined by introduction rules.
4. Equality Rule
   This rule shows how a function defined by means of elimination rules operates on the canonical elements of the type which are generated by the introduction rules.

However, the formation rules show how the type is constructed. So we can replace these rules with the syntax definition on EUODHILOS. The equality rules show how the elements of the type are transformed. Hence we can replace them with the rewriting rules. We define other introduction and elimination rules as ordinary inference rules.

Martin-Löf introduced so many types, here we define only the following types.

1. Cartesian Product: $(\Pi x \in A)B(x)$
   This type corresponds to a proposition $(\forall x \in A)B(x)$. When $B(x)$ does not depend on x, this type means a type of $A \rightarrow B$ and corresponds to a proposition $A \supset B$.

2. Disjoint Union of a Family of Types: $(\Sigma x \in A)B(x)$
   This type corresponds to a proposition $(\exists x \in A)B(x)$. When $B(x)$ does not depend on x, this type means a type of $A \times B$ and corresponds to a proposition $A \wedge B$.

3. Disjoint Union of Two Types: $A + B$
   This type corresponds to a proposition $A \vee B$.

4. Type of Integers: N
   This type corresponds to a proposition $N(x)$ which means x is an integer.

3.1 Language system of constructive type theory

Object language

```
judgment --> term,epsilon,type;
judgment --> exp,epsilon,type;
judgment --> type,equal,type;
judgment --> meta_judgment;
```

```
judgment --> meta_judgment,"(",type,")";
judgment --> meta_judgment,"(",term,")";

exp --> term,equal,term;
epsilon --> "∈";

variable --> meta_variable;

pure_term --> bind_op,variable,".",pure_term;
pure_term --> pure_term1;
pure_term1 --> pure_term1,plus,pure_term2;
pure_term1 --> pure_term2;
pure_term2 --> pure_term2,multiply,pure_term3;
pure_term2 --> pure_term3;
pure_term3 --> "(",pure_term,")";
pure_term3 --> basic_term;

basic_term --> "0";
basic_term --> variable;
basic_term --> meta_term;
basic_term --> unary_op,"(",term,")";
basic_term --> binary_op,"(",term,",",term,")";
basic_term --> ternary_op,"(",term,",",term,",",term,")";

plus --> "+";
multiply --> "*";
unary_op --> "sn"|"car"|"cdr"|"inl"|"inr"|meta_term;
binary_op --> "app"|"cons"|"E"|meta_term;
ternary_op --> "D"|"R"|meta_term;
bind_op --> "λ";

term --> pure_term;
term --> pure_term,"/",pure_term;

pure_type --> pure_type,imply,pure_type1;
pure_type --> pure_type1;
pure_type1 --> pure_type1,or,pure_type2;
pure_type1 --> pure_type2;
pure_type2 --> pure_type2,and,pure_type3;
pure_type2 --> pure_type3;
pure_type3 --> not,pure_type3;
pure_type3 --> "(",pure_type3,")";
pure_type3 --> "(",type_bind_op,variable,"∈",pure_type,")",pure_type3;
pure_type3 --> basic_type;

basic_type --> "⊥"|"N";
basic_type --> meta_type;
basic_type --> meta_type,"(",term,")";
basic_type --> meta_type,"(",term,",",term,")";
basic_type --> meta_type,"(",term,",",term,",",term,")";

type --> pure_type;
```

```
type --> pure_type,"/",pure_type;

equal --> "=";
imply --> "⊃";
or --> "∨";
and --> "∧";
not --> "∼";
type_bind_op --> "∀"|"∃";
```

Meta language

```
meta_type --> "A"-"C"|"P"|"Q"|"S"-"Z";
meta_term --> "a"-"h"|"p"-"t"|"s1"|"s2"|"s3"|"t1"|"t2"|"t3";
meta_variable --> "u"-"z"|"u1"|"u2"|"u3"|"v1"|"v2"|"v3"|"w1"|"w2"|"w3";
meta_judgment --> "J"|"J1"|"J2"|"J3";
```

Operator definition

```
operator
    multiply; plus; "/"; bind_op; not; type_bind_op; and; or; imply;
    equal; epsilon;
predicate
    unary_op, binary_op, ternary_op, meta_term, meta_type, meta_judgment.
```

3.2 Derivation system of constructive type theory

Axioms

$$A = A \quad (= \texttt{ref(type)})$$

$$0 \in \texttt{N} \quad (\texttt{NI})$$

`=ref(type)` is originally an inference rule with an assumption that A is a type, like $\dfrac{A\ type}{A = A}$. However in this case, we do not need this assumption since we define A as `meta_type` and A is implicitly identified as a type. Hence we can define this as an axiom.

Though Martin-Löf define (`NI`) as an introduction rule of `N`, we define this as an axiom since it has no assumption.

Inference Rules

$$\frac{a \subset A}{a = a \in A}\ (= \texttt{ref(term)}) \qquad \frac{a = b \in A}{b = a \in A}\ (= \texttt{sym(term)})$$

$$\frac{a = b \subset A \quad b = c \in A}{a = c \in A}\ (= \texttt{tra(term)})$$

$$\frac{A = B}{B = A}\ (= \mathtt{sym(type)}) \qquad \frac{A = B \quad B = A}{A = C}\ (= \mathtt{tra(type)})$$

$$\frac{a = b \in A \quad J(b/a)}{J(a)}\ (\mathtt{subst(term)}) \qquad \frac{A = B \quad J(B/A)}{J(A)}\ (\mathtt{subst(type)})$$

$$\frac{a \in A \quad b \in B}{\mathtt{cons}(a,b) \in A \wedge B}\ (\wedge\mathtt{I}) \qquad \frac{c \in A \wedge B}{\mathtt{car}(c) \in A}\ (\wedge\mathtt{E1}) \qquad \frac{c \in A \wedge B}{\mathtt{cdr}(c) \in B}\ (\wedge\mathtt{E2})$$

$$\frac{a \in A}{\mathtt{inl}(a) \in A \vee B}\ (\vee\mathtt{I1}) \qquad \frac{b \in B}{\mathtt{inr}(b) \in A \vee B}\ (\vee\mathtt{I2})$$

$$\frac{c \in A \vee B \quad \begin{array}{c}[a \in A]\\ \vdots \\ d(a/x) \in C(\mathtt{inl}(a)/c)\end{array} \quad \begin{array}{c}[b \in B]\\ \vdots \\ e(b/y) \in C(\mathtt{inr}(b)/c)\end{array}}{\mathtt{D}(c, \lambda x.d(x), \lambda y.e(y)) \in C(c)}\ (\vee\mathtt{E})$$

$$\frac{\begin{array}{c}[x \in A]\\ \vdots \\ f(x) \in B\end{array}}{\lambda x.f(x) \in A \supset B}\ (\supset\mathtt{I}) \qquad \frac{f \in A \supset B \quad a \in A}{\mathtt{app}(f,a) \in B}\ (\supset\mathtt{E})$$

$$\frac{\begin{array}{c}[y \in A]\\ \vdots \\ b(y/x) \in B(y/x)\end{array}}{\lambda x.b(x) \in (\forall x \in A)B(x)}\ (\forall\mathtt{I}) \qquad \frac{c \in (\forall x \in A)B(x) \quad a \in A}{\mathtt{app}(c,a) \in B(a/x)}\ (\forall\mathtt{E})$$

$$\frac{a \in A \quad b \in B(a/x)}{\mathtt{cons}(a,b) \in (\exists x \in A)B(x)}\ (\exists\mathtt{I}) \qquad \frac{c \in (\exists x \in A)B(x) \quad \begin{array}{c}[\mathtt{cons}(a,b) \in A \wedge B(a/x)]\\ \vdots \\ d(a/x, b/y) \in C(\mathtt{cons}(a,b)/c)\end{array}}{\mathtt{E}(c, \lambda x.\lambda y.d(x,y)) \in C(c)}\ (\exists\mathtt{E})$$

$$\frac{t \in \mathtt{N}}{\mathtt{sn}(t) \in \mathtt{N}}\ (\mathtt{NI}) \qquad \frac{c \in \mathtt{N} \quad d \in C(0/c) \quad \begin{array}{c}[\mathtt{cons}(a,b) \in \mathtt{N} \wedge C(a/c)]\\ \vdots \\ e(a/x, b/y) \in C(\mathtt{sn}(a)/c)\end{array}}{\mathtt{R}(c, d, \lambda x.\lambda y.e(x,y)) \in C(c)}\ (\mathtt{NE})$$

The original rule of ($\exists$E) has the following form:

$$\frac{c \in (\exists x \in A)B(x) \quad \begin{array}{c}[a \in A, \quad b \in B(a/x)]\\ \vdots \\ d(a/x, b/y) \in C(\mathtt{cons}(a,b)/c)\end{array}}{\mathtt{E}(c, \lambda x.\lambda y.d(x,y) \in C(c))}\ (\exists\mathtt{E})$$

However, because of the restriction that we must put only one discharged assumption on a premise, we define the rule ($\exists$E) as above. Similarly for (NE).

Rewriting Rules

$$\frac{\mathtt{app}(\lambda x.a(x), b)}{a(b/x)}\ (\mathtt{beta}) \qquad \frac{\mathtt{cons}(\mathtt{car}(t), \mathtt{cdr}(t))}{t}\ (\mathtt{cons})$$

$$\frac{\mathtt{car}(\mathtt{cons}(s,t))}{s}\ (\mathtt{cons\text{-}car}) \qquad \frac{\mathtt{cdr}(\mathtt{cons}(s,t))}{t}\ (\mathtt{cons\text{-}cdr})$$

$$\frac{\mathtt{D}(\mathtt{inl}(a), \lambda x.d(x), \lambda y.e(y))}{d(a/x)}\ (\mathtt{D1}) \qquad \frac{\mathtt{D}(\mathtt{inr}(b), \lambda x.d(x), \lambda y.e(y))}{e(b/y)}\ (\mathtt{D2})$$

$$\frac{\mathtt{E}(\mathtt{cons}(a,b), \lambda x.\lambda y.d(x,y))}{d(a/x, b/y)}\ (\mathtt{E})$$

$$\frac{\mathtt{R}(0, d, \lambda x.\lambda y.e(x,y))}{d}\ (\mathtt{R1}) \qquad \frac{\mathtt{R}(\mathtt{sn}(a), d, \lambda x.\lambda y.e(x,y))}{e(a/x, \mathtt{R}(a, d, \lambda x.\lambda y.e(x,y))/y)}\ (\mathtt{R2})$$

$$\frac{\sim A}{A \supset \bot}\ (\sim \mathtt{def})$$

$$\frac{\mathtt{car}(t)}{\mathtt{E}(t, \lambda x.\lambda y.x)}\ (\mathtt{car}) \qquad \frac{\mathtt{cdr}(t)}{\mathtt{E}(t, \lambda x.\lambda y.y)}\ (\mathtt{cdr})$$

$$\frac{a+b}{\mathtt{R}(b, a, \lambda x.\lambda y.\mathtt{sn}(y))}\ (+\mathtt{def}) \qquad \frac{a*b}{\mathtt{R}(b, 0, \lambda x.\lambda y.y+a)}\ (*\mathtt{def})$$

3.3 Proof examples

Figure 2 shows three proofs of the following simple theorems and derived rule:

1. $(\forall x \in A)(B(x) \supset C(x)) \supset ((\forall x \in A)B(x) \supset (\forall x \in A)C(x))$ is true.
2. $(\exists x \in A)B(x) \wedge (\exists x \in A)C(x) \supset (\exists x \in A)(B(x) \wedge C(x))$ is true.
3. If $a \in N$ and $b \in N$ then $a + b \in N$.

Figure 3 shows the screen layout of the proof of "$\sim\sim (A \vee \sim A)$ is true" and the related inference and rewriting rules.

Figure 4 shows the axiom of choice is true:

$$(\forall x \in A)(\exists y \in B(x))C(x,y) \supset (\exists y \in (\forall x \in A)B(x))(\forall x \in A)C(x, app(y,x))$$

has a proof. In this case, $(\forall x \in A)B(x)$ is equivalent to $(\Pi x \in A)B(x)$ as a type.

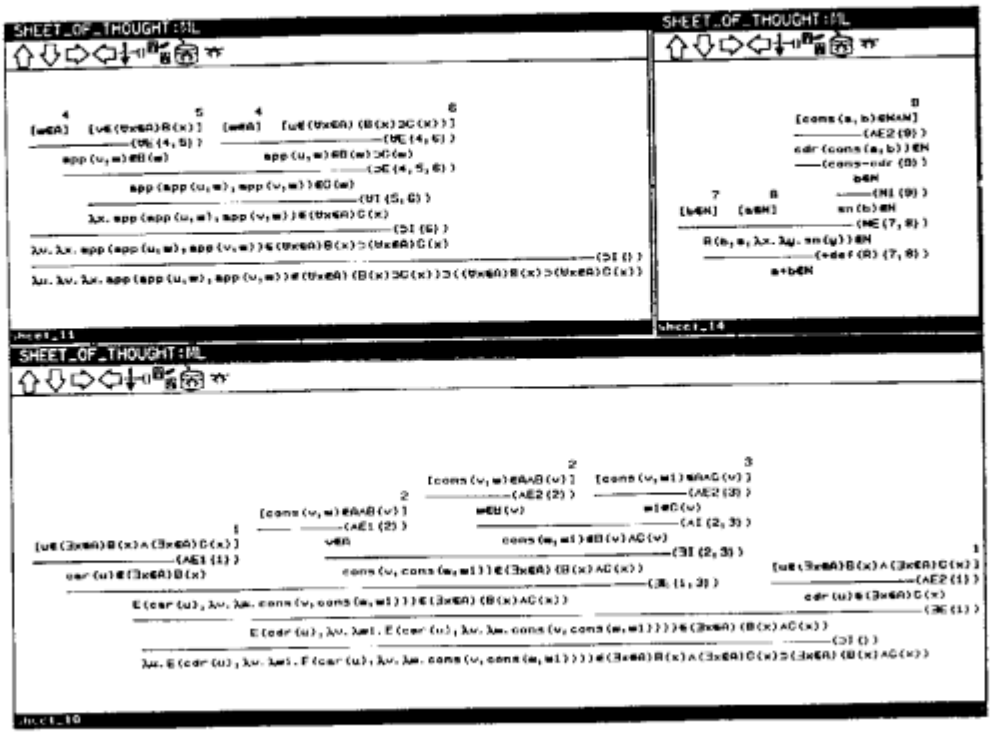


Figure 2: Some simple proofs of constructive type theory

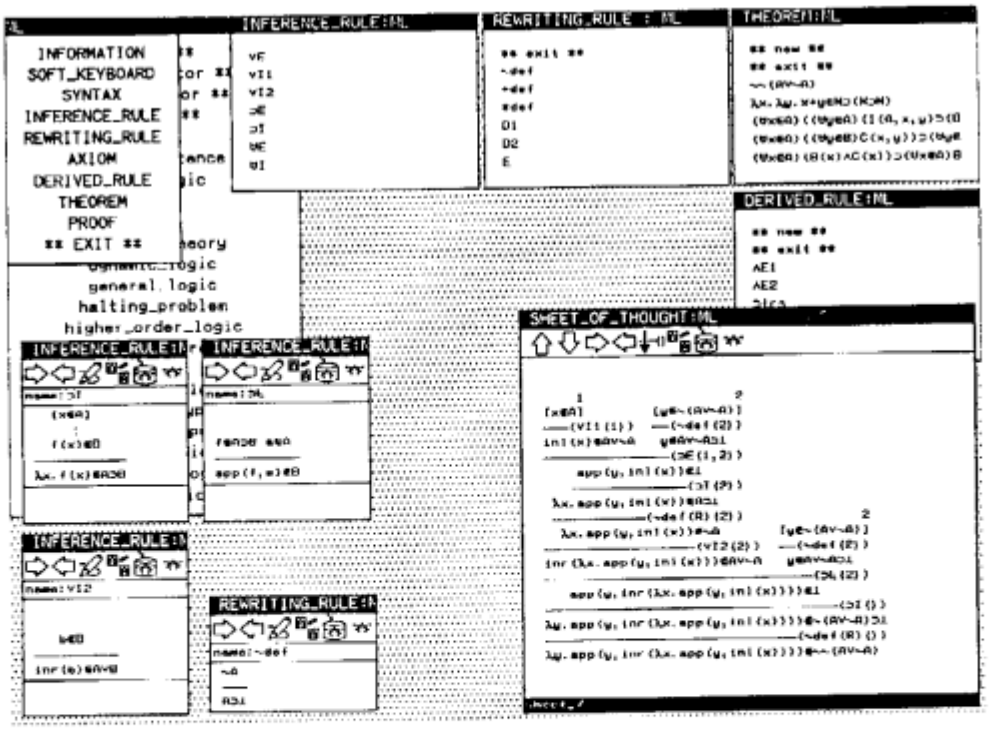


Figure 3: A proof of "$\sim\sim(A \vee \sim A)$ is true"

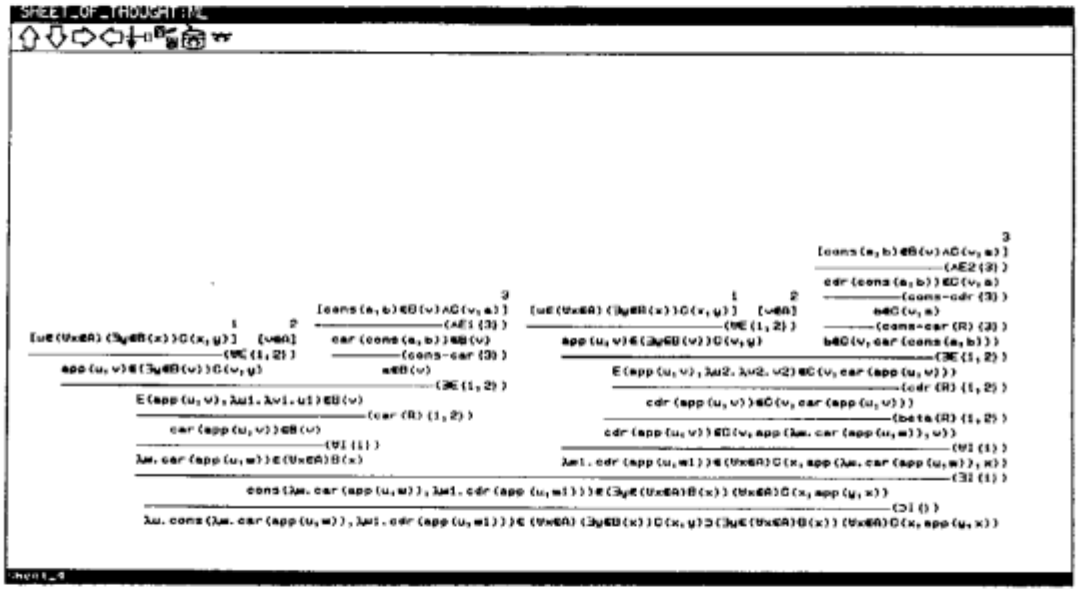


Figure 4: A proof of the axiom of choice

4 Hoare logic

Hoare logic [Hoare 69] 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. The inference rules of Hoare logic can be described in a Hilbert-style presentation.

4.1 Language system of Hoare logic

Syntax of object language

```
h-formula --> formula, "{", program, "}", formula ;
formula --> formula, "⊃", formula1 ;
formula --> formula1 ;
formula1 --> formula1, "∨", formula2;
formula1 --> formula2 ;
formula2 --> formula2, "∧", formula3 ;
formula2 --> formula3 ;
formula3 --> "(", formula, ")" ;
formula3 --> "~", formula3 ;
formula3 --> basic_formula;
basic_formula --> "true" | term, "=", term ;

term --> variable | constant | "(",term,")" | term, "+", term |
      term, "*", term | term, "!";
variable --> "x" | "y" |"z";
constant --> "1" | "0" ;

program --> program, ";", program1;
program --> program1;
program1 --> assignment_statement |
      "while", formula, "do", program, "od" |
      "if", formula, "then", program, "else", program, "fi" |
      "(", program,")";
assignment_statement --> variable, ":=", term ;
```

Syntax of meta language

```
meta_program --> "A" | "B";
meta_var --> "X" | "Y" | meta_var, "/", term;
meta_term --> "T" ;
meta_formula --> "P" | "E" | "F" |"G";
basic_formula --> meta_var;
term --> meta_term;
variable --> meta_var;
program --> meta_program;
program1 --> meta_program.
```

Operator definition

operator
"!"; "$*$"; "+"; "=" ; "$\sim$"; "$\wedge$"; "$\vee$"; "$\supset$"; ":=" ; "while" ; "if" ; "{".

4.2 Derivation system of Hoare logic

Axiom

$P(T)\{X := T\}P(X/T)$ (Assignment axiom)

Rewriting rule

$$\frac{z = y!}{z \times (y+1) = (y+1)!}$$ (Arithmetic rule)

Inference rules

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

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

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

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

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

4.3 Partial correctness proof of a program

Figure 5 shows the screen layout of the proof of the following partial correctness assertion of a factorial program:

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

with the precondition "*true*" and postcondition "$z = x!$".

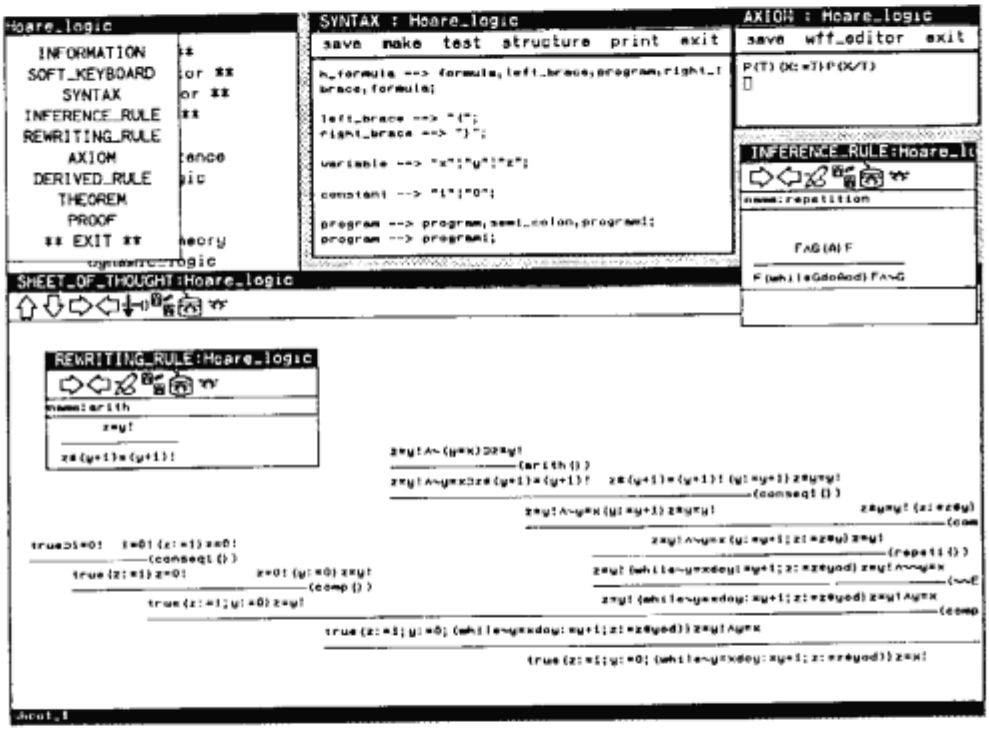


Figure 5: Hoare logic and partial correctness proof of a program

5 Dynamic logic

Dynamic logic [Harel 84] 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 $\langle a\rangle p$, 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 inference rules of dynamic logic can be described in a Hilbert-style presentation.

5.1 Language system of dynamic logic

Syntax of object language

```
dynamic_formula --> "⟨", regular_program,"⟩", formula3 ;
dynamic_formula --> "[", regular_program, "]", formula3 ;

formula --> formula, "≡", formula0 ;
formula --> formula0 ;
formula0 --> formula0, "⊃", formula1 ;
formula0 --> formula1 ;
formula1 --> formula1, "∨", formula2;
formula1 --> formula2 ;
formula2 --> formula2, "∧", formula3 ;
formula2 --> formula3 ;
formula3 --> "(", formula, ")" ;
formula3 --> "∼" , formula3 ;
formula3 --> dynamic_formula ;
formula3 --> "true" ;
formula3 --> term, "=", term ;
formula3 --> term, ">", term ;
formula3 --> term, "≥", term ;

term --> variable | constant ;
term --> term, "+", term | term, "−", term | term, "×", term | term, "!" |
      "(", term, ")" | term, "/", term ;
variable --> "x" | "y" |"z" | "n" ;
constant --> "1" | "0" ;

regular_program --> regular_program, ";", regular_program1 ;
regular_program --> regular_program,"|", regular_program1;
regular_program --> regular_program1 ;
regular_program1 --> regular_program2, "*";
regular_program1 --> regular_program2 ;
regular_program2 --> assignment_statement ;
regular_program2 --> formula, "?";
regular_program2 --> "(", regular_program, ")" ;
assignment_statement --> variable, ":=", term ;
```

Syntax of meta language

```
regular_program2 --> meta_program ;
formula3 --> meta_formula, "(", meta_formula, ")" ;
formula3 --> meta_formula ;
term --> meta_term ;
variable --> meta_variable ;
meta_term --> meta_variable ;
meta_program --> "A" | "B" ;
meta_variable --> "X" ;
meta_formula --> "P" | "Q" | "R" | "S".
```

Operator definition

```
operator
    "/"; "!"; "×"; ("+", "−"); ("⟨", "["); ("∼", ">", "≥", "="); "∧"; "∨"; "⊃";
    "≡"; ("?", ":="); "*"; (";", "|");
predicate
    meta_formula.
```

5.2 Derivation system of dynamic logic

Axioms

1. $[Q?]P \equiv (Q \supset P)$ (test)
2. $[X := T]P(X) \equiv P(T/X)$ (assignment axiom)
3. $[A; B]P \equiv [A][B]P$ (composition)
4. $\langle A; B\rangle P \equiv \langle A\rangle\langle B\rangle P$ (composition)
5. $[A|B]P \equiv ([A]P \wedge [B]P)$ (nondeterministic selection)

Theorems

1. $\langle (x = 0)?\rangle true \equiv (x = 0 \supset true)$
2. $n \geq 0 \wedge x = n + 1 \supset \langle (x > 0)?\rangle (x = n + 1)$
3. $x = n + 1 \supset \langle z := x \times z\rangle (x = n + 1)$
4. $x = n + 1 \supset \langle x := x - 1\rangle (x = n)$
5. $z \times x! = n! \wedge x > 0 \supset [z := x \times z](z \times (x - 1)! = n!)$
6. $z \times (x - 1)! = n! \supset [x := x - 1](z \times x! = n!)$
7. $x = n \supset [z := 1](z \times x! = n!)$
8. $z \times x! = n! \supset [(x = 0)?](z = n!)$

Rewriting rules

$$\frac{[A]P}{\sim\langle A\rangle\sim P}\ \text{(def)}$$

$$\frac{\sim P\equiv\sim Q}{P\equiv Q}\ \text{(neg-elim)}$$

$$\frac{\sim\sim P}{P}\ \text{(double-neg-elim)}$$

$$\frac{n\geq 0\wedge x=n}{x\geq 0}\ \text{(arithmetic)}$$

$$\frac{true\wedge P}{P}\ \text{(true-elim)}$$

Inference rules

$$\frac{P\quad P\supset Q}{Q}\ \text{(modus ponens)}$$

$$\frac{P\supset Q}{[A]P\supset[A]Q}\ \text{(necessitation)}$$

$$\frac{P\supset[A]P}{P\supset[A*]Q}\ \text{(invariance)}$$

$$\frac{n\geq 0\wedge P(n+1)\supset\langle A\rangle P(n)}{n\geq 0\wedge P(n)\supset\langle A*\rangle P(0)}\ \text{(convergence)}$$

$$\frac{P\supset\langle A\rangle Q\quad Q\supset\langle B\rangle R}{P\supset\langle A;B\rangle R}\ \text{(composition 1)}$$

$$\frac{P\supset[A]Q\quad Q\supset[B]R}{P\supset[A;B]R}\ \text{(composition 2)}$$

$$\frac{P\supset\langle A\rangle Q\quad R\supset\langle A\rangle S}{P\wedge R\supset\langle A\rangle(Q\wedge R)}\ \text{(derived-rule1)}$$

$$\frac{P(Q)\quad Q\equiv R}{P(R/Q)}\ \text{(replacement)}$$

$$\frac{P\equiv Q}{Q\equiv P}\ \text{(symmetricity)}$$

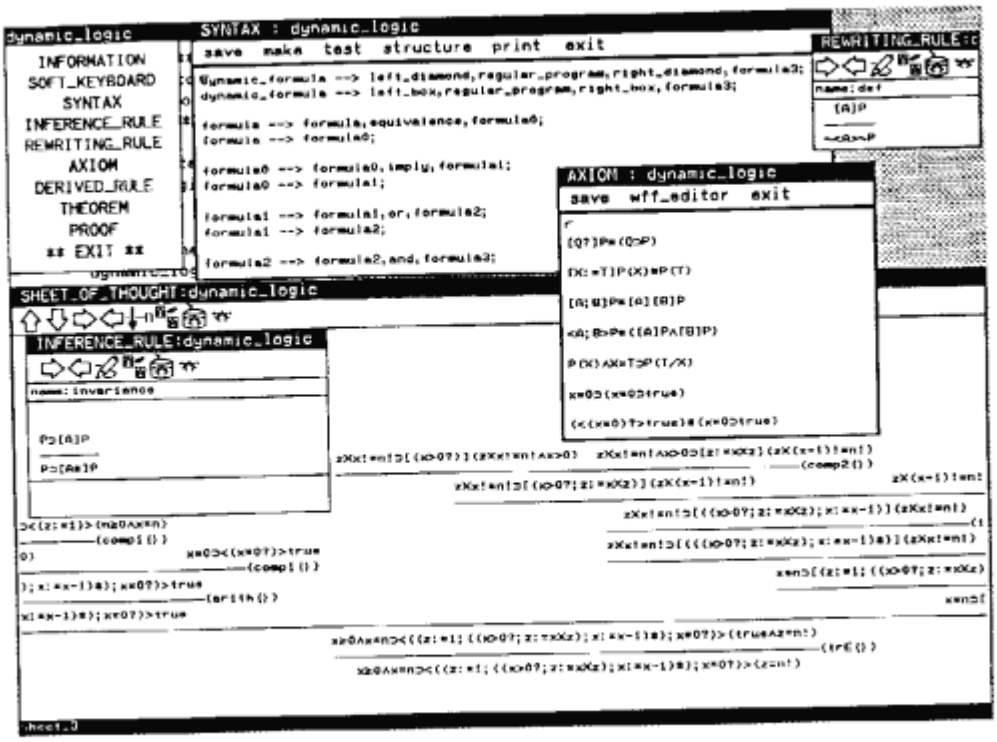


Figure 6: Dynamic logic and Partial correctness proof of a program

5.3 Reasoning about programs

Figure 6 shows the screen layout of the proofs of 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 : $x \geq 0 \wedge x = n \supset$
$\langle z := 1; ((x > 0)?; z := x \times z; x := x - 1)*; (x = 0)?\rangle (z = n!)$

6 Intensional logic

Intensional logic [Gallin 75] 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.

6.1 Language system of intensinal logic

Meta language

```
meta_formula --> pred-const, "(", term(_), ")" ;
meta_formula --> meta_formula, "=>", meta_formula ;
pred_const --> "beweis" ;
meta_formula --> meta_term(_) ;
meta_variable --> "X" | "Y" ;
meta_term(_) --> "R" | "S" | "A" | "B" | "P" | "F" ;
meta_term(_) --> meta_variable ;
meta_term(_) --> meta_term(_), ":", type(_) ;
meta_type(_) --> "a" | "b" | "c" | "T" | "T1" | "T2" | "T3" ;
```

Object language

```
term(t) --> term(t), "⊃", term1(t) ;
term(T) --> term1(T) ;
term1(t) --> term1(t), "∨", term2(t) ;
term1(T) --> term2(T) ;
term2(t) --> term2(t), "∧", term3(t) ;
term2(T) --> term3(T) ;
term3(t) --> term3(T), "=", term7(T) ;
term3(T) --> term7(T) ;
term7(T2) --> term7((s,(T1,T2))), "{", term(T1), "}" ;
term7(T) --> term4(T) ;
term4(t) --> bind_op, variable(T), ".", term5(t) ;
bind_op --> "∀" | "∃" ;
term4(T) --> term5(T)
term4((T1,T2)) --> bind_op, variable(T1), ".", term5(T2) ;
bind_op --> "λ" ;
term5(t) --> "∼", term5(t) ;
term5(T2) --> term5((T1,T2)), "•", term6(T1) ;
term5(T) --> term6(T) ;
term6((s,T)) --> "^", term6(T) ;
term6(T) --> "ˇ", term6((s,T)) ;
term6(t) --> "□", term6(t) ;
term6(t) --> "◇", term6(t) ;
term6(T) --> "(", term(T), ")" ;
term6(T) --> variable(T) | constant(T);
term6(T) --> meta_term2(T), "(", term(_), ")" ;
term6(T) --> meta_term2(T), "(", term(_), "/", term(_), ")" ;
meta_term2(T) --> meta_term(T) ;
term6(T) --> meta_term(T) ;
```

```
variable(T) --> var_sym, ":", type(T) | meta_variable, ":", type(T) ;
constant(t) --> truth_value, ":", type(t) ;
constant (T)--> const_sym, ":", type(T) ;
truth_value --> "true" | "false";
var_sym --> "x" | "y" | "p" ;
const_sym --> "fish" | "believe" | "walk" | "j" ;

type(e) --> "e" ;
type(t) --> "t" ;
type(T) --> meta_type(_) ;
type((T1,T2)) --> "(", type(T1), "," , type(T2), ")" ;
type((s,T)) --> "(", "s", "," , type(T), ")" ;
```

Operator definition

```
operator
    "/"; (":", ","); ("^", "v", "□", "◇"); ("•", "∼"); "{"; bind_op; "=" ;
    "∧"; "∨" ; ("⊃", "=>") ;
predicate
    meta_term2, pred_const.
```

6.2 Derivation system of intensional logic

Axioms

1. $G : (t,t) \bullet true : t \wedge G : (t,t) \bullet false : t = \forall X : t.G : (t,t) \bullet X : t$
2. $X : a = Y : a \supset F : (a,t) \bullet X : a = F : (a,t) \bullet Y : a$
3. $\forall X : a.(F : (a,b) \bullet X : a = G : (a,b) \bullet X : a) = (F : (a,b) = G : (a,b))$
4. $(\lambda X : a.A(X : a)) \bullet B = A(B)$
5. $\Box(^{\vee}F : (s,a) =^{\vee} G : (s,a)) = (F : (s,a) = G : (s,a))$
6. $^{\vee\wedge}A : a = A : a$

Theorems

1. $(P : t = true : t) = P : t$
2. $\lambda X : a.Q : b = \lambda X : a.Q : b$

Inference rules

Meta-Rule

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

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

$$\frac{\begin{array}{c}A\\ \vdots \\ B\end{array}}{A => B}\ (=>\ \text{I})$$

Object-Rule (IL Rule)

$$\frac{A(R) \quad R = S}{A(S/R)} \text{ (Replace)}$$

$$\frac{R = S}{S = R} \text{ (Symmetricity)}$$

Rewriting rules

$$\frac{\lambda X : a.P : t = \lambda X : a.true : t}{\forall X : a.P : t} \text{ (}\forall\text{-Definition)}$$

$$\frac{A\{R\}}{(^{\vee}A) \bullet R} \text{ (Brace convention)}$$

$$\frac{F \bullet G}{F(G)} \text{ (Notational convention)}$$

6.3 Reflective proof and Montague's semantics

The following metatheorem is ingeniously proved with the help of the reflection principle.

$$\vdash P : t \Rightarrow \vdash \forall x : a.P : \bar{t} \text{ (Generalization rule)}$$

See Figure 7 for the screen layout of the proof.

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

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

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 :

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

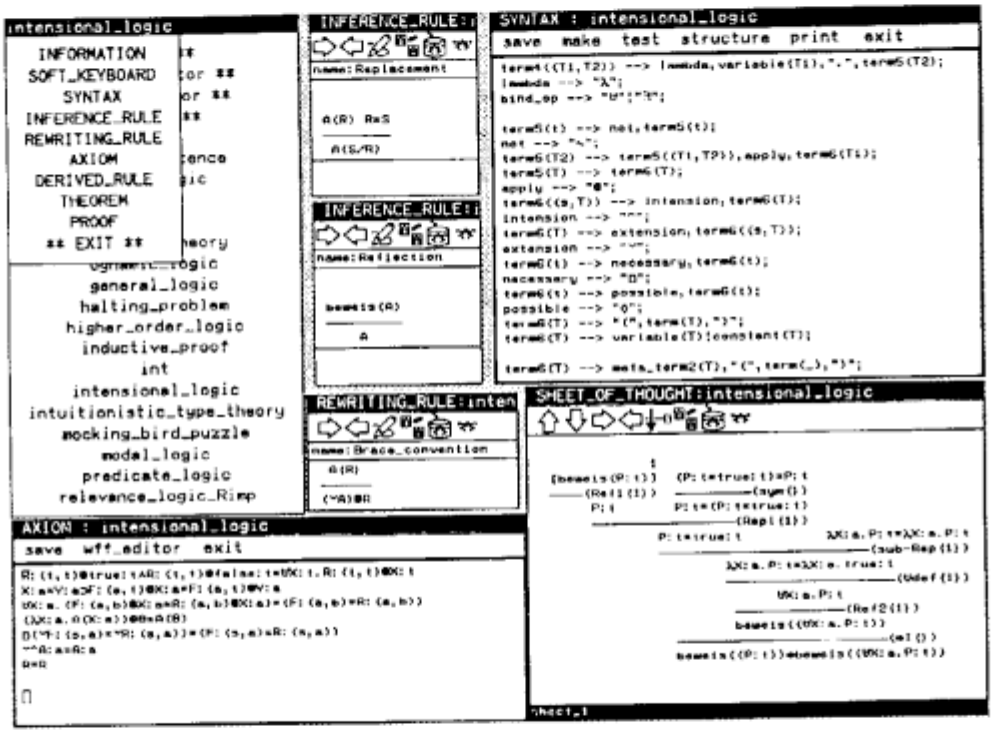


Figure 7: Intensional logic and a reflective proof

7 General logic

General logic is a formal system which yields a unified account of a fairly wide range of logical systems. Diverse logics are displayed as variations on a single theme [Slaney 90]. The general logic has been very successfully and smoothly handled on a general reasoning system EUOD-HILOS by specifying those variations on a single theme as rewriting rules (See [Sawamura 91b] for the details).

7.1 Language system of general logic

Object language

```
sequent --> bunch, ":", formula;

bunch --> bunch, ",", bunch1 | bunch1;   %pooling
bunch1 --> bunch1, ";", bunch2 | bunch2;  %application
bunch2 --> formula;
bunch2 --> "(", bunch, ")";

formula --> formula1, "->", formula | formula1;
formula1 --> formula1, "∨", formula2 | formula2;
formula2 --> formula2, "&", formula3 | formula3;
formula3 --> bind_op, variable, ".", formula3 | formula4;
bind_op --> "∀" | "∃";
variable --> "v" | "u" | "c" | "d";
formula4 --> "∼", formula4 | formula5;
formula5 --> "(", formula, ")" ;
formula5 --> "p" | "q" | "r" | "true";
formula5 --> predicator, "(", variable, ")";
predicator --> "g" | "h" | "k";
```

Meta language

```
meta_bunch --> "X" | "Y" | "Z";
meta_bunch1 --> meta_form, "(", meta_arg, ")";
meta_arg --> meta_bunch | meta_form;
meta_form --> "A" | "B" | "C" | "Γ" |
      meta_form, "(", meta_var, ")" |
      meta_form, "(", meta_term, ")" |
      meta_form, "(", substitution, ")" |
      meta_form, "(", meta_form1, ")" ;
meta_form1 --> meta_form;
substitution --> meta_var, "/", meta_var, ")" |
      meta_term, "/", meta_var |
      meta_form1, "/", meta_form1 |
      meta_bunch, "/", meta_form1 |
      meta_bunch, "/", meta_bunch;
meta_var --> "a" | "x" | "y" | "z";
meta_term --> "t";
```

Interface between object and meta languages

```
bunch2 --> meta_bunch | meta_bunch1;
formula5 --> meta_form;
variable --> meta_var.
```

Operator precedence

```
operator
    "/"; "~"; bind_op; "&"; "∨"; "->"; ";"; ","; ":";
predicate
    meta_form, predicator.
```

7.2 Derivation system of general logic

Axiom

$$A : A$$

Rewriting rules

$$\frac{(X,(Y,Z))}{(X,Y,Z)}\ (\text{SR1\_1}) \qquad \frac{(X,Y,Z)}{(X,(Y,Z))}\ (\text{SR1\_2})$$

$$\frac{(X,Y)}{(Y,X)}\ (\text{SR2})$$

$$\frac{(X,X)}{X}\ (\text{SR3\_1}) \qquad \frac{X}{(X,X)}\ (\text{SR3\_2})$$

$$\frac{X}{(X,Y)}\ (\text{SR4})$$

$$\frac{true;X}{X}\ (\text{SR5\_1}) \qquad \frac{X}{true;X}\ (\text{SR5\_2})$$

$$\frac{X;(Y;Z)}{(X;Y);Z}\ (\text{SR6})$$

$$\frac{X;Y}{Y;X}\ (\text{SR7})$$

$$\frac{X;X}{X}\ (\text{SR8})$$

$$\frac{X}{X;Y}\ (\text{SR9})$$

Inference rules

$$\frac{X : A\&B}{X : A}\ (\&\text{E\_1}) \qquad \frac{X : A\&B}{X : B}\ (\&\text{E\_2})$$

$$\frac{X : A \quad Y : B}{(X,Y) : A\&B}\ (\&\text{I})$$

$$\frac{X : A \mathrel{->} B \quad Y : A}{X;Y : B}\ (\text{MPP})$$

$$\frac{X;A : B}{X : A \mathrel{->} B}\ (\text{CP})$$

$$\frac{X : A \mathrel{->} B \quad Y :\sim B}{X;Y :\sim A}\ (\text{MTT})$$

$$\frac{X : A}{X :\sim\sim A}\ (\text{DNI}) \qquad \frac{X :\sim\sim A}{X : A}\ (\text{DNE})$$

$$\frac{X : A}{X : A \vee B}\ (\vee\ \text{I\_1}) \qquad \frac{X : B}{X : A \vee B}\ (\vee\ \text{I\_2})$$

$$\frac{X : A \vee B \quad \Gamma(A) : C \quad \Gamma(B/A) : C}{\Gamma(X/A) : C}\ (\vee\ \text{E})$$

$$\frac{X : A(a)}{X : \forall x A.(x/a)}\ (\text{UI})\ \ a\text{: eigenvariable and } x \text{ is free for } a \text{ in } A.$$

$$\frac{X : \forall x.A(x)}{X : A(t/x)}\ (\text{UE})\ \ t \text{ is free for } x \text{ in } A(x).$$

$$\frac{X : A(t)}{X : \exists x.A(x/t)}\ (\text{EI})\ \ x \text{ is free for } t \text{ in } A(t).$$

$$\frac{X : \exists x.A(x) \quad \Gamma(A(a/x)) : B}{\Gamma(X/A) : B}\ (\text{EE})\ \ a\text{: eigenvariable, } a \notin B \text{ and } a \notin \Gamma.$$

Then a logic hierarchy in the general logic is as follows:

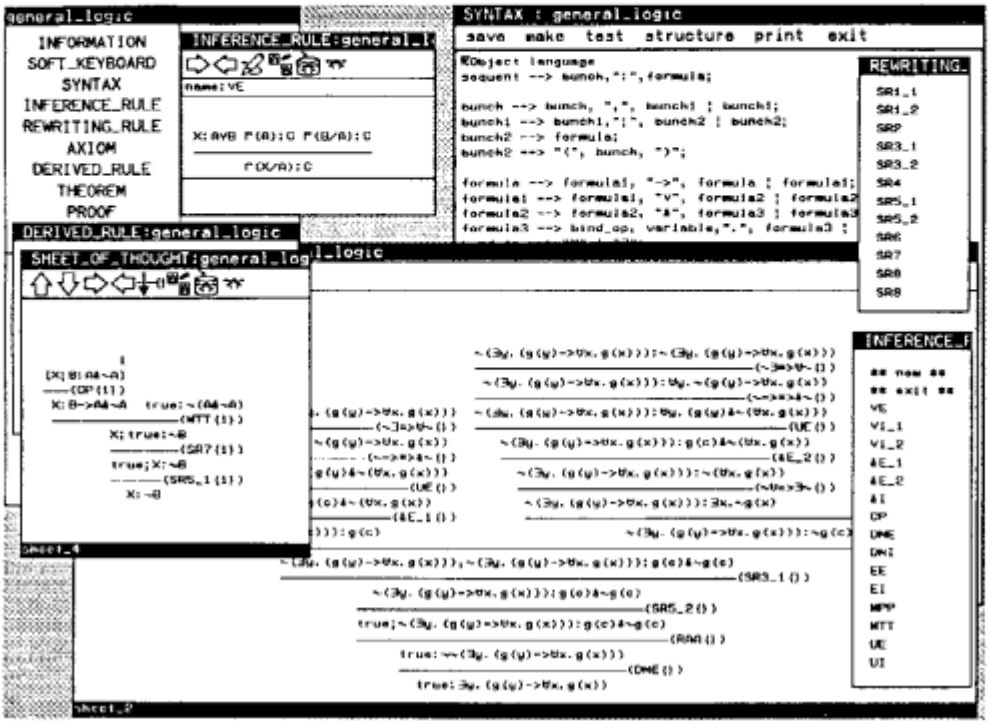
DW = {SR1-SR5}
C (RW,R-W) = **DW** ∪ {SR6, SR7}
R = **C** ∪ {SR8} (Relevance logic)
K = **R** ∪ {SR9} (Classical logic)
CK = **C** ∪ {SR9}.

7.3 Proof examples

The proof examples in the various systems covered in the general logic include:

$true : (\sim p \rightarrow q) \rightarrow (\sim q \rightarrow p)$
$p \rightarrow q, p \rightarrow r : p \rightarrow q \& r$
$p, q \vee r : p \& q \vee r$ (Distribution)
$true : \sim (A \& \sim A)$
$X; B : A \& \sim A \vdash X : \sim B$ (Reductio ad absurdum)
$true : \exists y.(g(y) \rightarrow \exists x.g(x))$ (Baffling formula).

See Figure 8 for the screen layout of the proofs of the first three theorems.



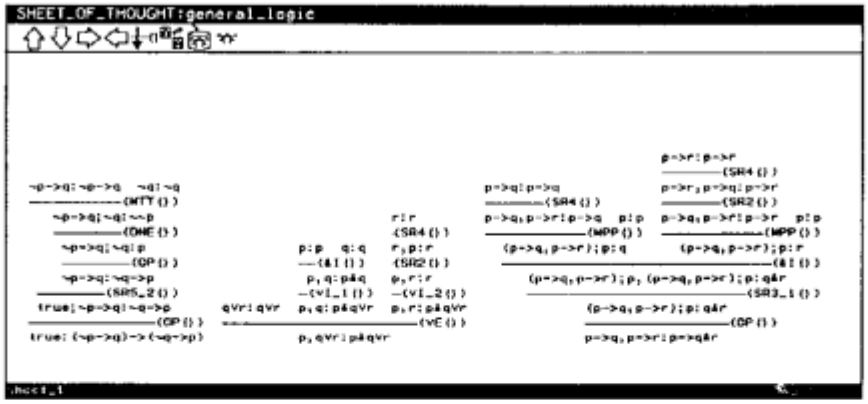


Figure 8: Proofs in the system DW

8 Relevance logic - Implicational calculus $R_{\rightarrow}$ based on tag calculus

Tag is an apparatus which controls inferences. In a natural deduction style presentation of first-order predicate calculus, it plays a role of the justification of an inference which is to be calculated in a set-theoretic manner. Tag in an implicational calculus $R_{\rightarrow}$ ingeniously controls an inference so as to yield a conclusion relevantly from an antecedent. The following axiomatization of an implicational calculus $R_{\rightarrow}$ is due to Meyer [Meyer 90].

8.1 Language system of $R_{\rightarrow}$

Object language

```
tag_formula --> tag, "=>", formula ;
tag_formula --> "=>", formula ;

formula --> formula1, "->", formula ;
formula --> formula1 ;
formula1 --> "(", formula, ")" ;
formula1 --> "p" | "q" | "r" | ;

tag --> tag, "*", tag1 ;
tag --> tag1 ;
tag1 --> "(", tag, ")" ;
tag1 --> "a" | "b" | "c" |"d" ;
```

Meta language

```
meta_formula --> "P" | "Q" | "R" ;
meta_tag --> "T1" | "T2" | "T3" ;
```

Interface between object and meta languages

```
formula1 --> meta_formula ;
tag1 --> meta_tag ;
```

Operator definition

```
operator
      ("->" ; "*"); "=>".
```

8.2 Derivation system of $R_{\rightarrow}$

Inference rules

$$\frac{(T1 * T2) * T3 => P}{(T1 * T3) * T2 => P} \text{ (Tag Rule } C : C\alpha\gamma\beta = \alpha\beta\gamma)$$

$$\frac{T1 * (T2 * T3) => P}{T1 * T2 * T3 => P}\ (\text{Tag Rule } B : B\alpha\beta\gamma = \alpha(\beta\gamma))$$

$$\frac{T1 * T2 * T2 => P}{T1 * T2 => P}\ (\text{Tag Rule } W : W\alpha\beta = \alpha\beta\beta$$

$$\frac{T1 => P}{T1 => P}\ (\text{Tag Rule } I : I\alpha = \alpha)$$

$$\frac{T1 => P -> Q \quad T2 => P}{T1 * T2 => Q}\ (->\text{E})$$

$$\frac{\begin{array}{c}[T1 => P] \\ \vdots \\ T2 * T1 => Q\end{array}}{T2 => Q}\ (->\text{I1})$$

$$\frac{\begin{array}{c}[T1 => P] \\ \vdots \\ T1 => Q\end{array}}{P -> Q}\ (->\text{I2})$$

8.3 Proof examples

Figure 9 shows the screen layout of the proofs of the following typical theorems in relevance logic $R_{->}$:

1. $(P -> (Q -> R)) -> (Q -> (P -> R))$ (Permutation)
2. $(P -> Q) -> ((R -> P) -> (R -> Q))$ (Prefixing)
3. $(P -> (P -> Q)) -> (P -> Q)$ (Contraction)
4. $P -> P$ (Self-implication)

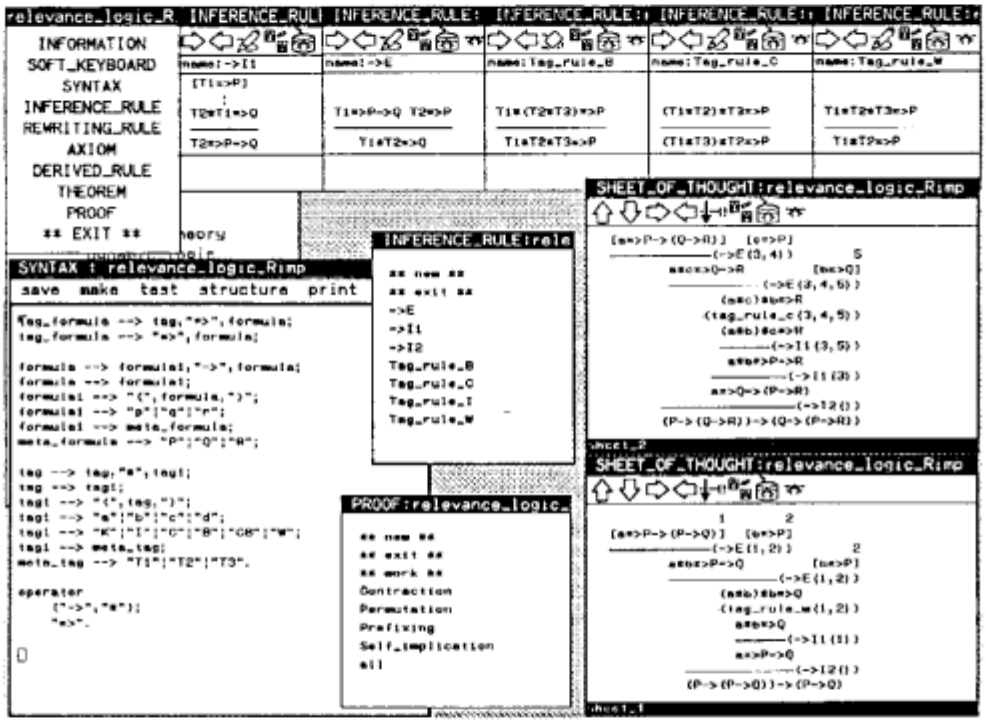


Figure 9: Proofs of relevant implication

9 Category Theory

Category theory[Mac Lane 71] was formulated as the theory of "naturalness." Now it becomes more and more popular as the abstract theory of mathematical structures represented by objects and arrows (or morphisms). It even has some applications in the computer science.

The following is a formulation for elementary theory of the categories. It is rather a tentative one, though. Some concepts have syntax definitions without having no corresponding derivational definitions.

9.1 Language system of category theory

In an ordinary formulation, a category consists of arrows and objects. In this formulation we chose arrows-only notation–objects are identified with the identity arrows one-to-one correspond to objects– because we think it is simpler to handle. There is another syntactic extension in the formulation. An arrow expression can be used as a formula, which will be read that the arrow exists.

Syntax of object language

```
formula --> formula1, "⇒", formula1 | formula1;
formula1 --> formula1, "∨", formula2 | formula2;
formula2 --> formula2, "∧", formula3 | formula3;
formula3 --> bind_op, variable, ".", formula3 | formula4;
formula4 --> "(", formula, ")"|
      arrow | arrow, "=", arrow | arrow_with_prop |
      meta_formula | meta_formula, "(", subst_arrow, ")";

arrow --> arrow1, arrow_op, arrow1 | arrow1;
arrow1 --> arrow_function, "(", arrow, ")"|
      arrow_function2, "(", arrow, ",", arrow, ")"|
      "(", arrow, ")"| arrow_constant |
      meta_arrow | meta_arrow_symbol, "(", subst_arrow, ")";
arrow_with_prop --> arrow, ":", property |
      arrow_with_prop, ":", property;

variable --> meta_variable;
subst_arrow --> meta_arrow | arrow, "/", meta_arrow;
meta_arrow --> meta_arrow_symbol;

property --> "monic"| "epic"| "iso"| "obj"|
      arrow, "--->", arrow | meta_property;

bind_op --> "∀"| "∃";
arrow_function --> "inv"| "dom"| "cod"| "0"| "1";
arrow_function2 --> "pr1"| "pr2";
arrow_constant --> "0"| "1";
arrow_op --> "."| "x"| "*";
```

Syntax of meta language

```
meta_variable --> "x"- "z"| "x1"| "y1"| "z1";
meta_formula --> "WFF"| "WFF1"| "WFF2"| "W"| "W1"| "W2";
meta_arrow_symbol --> "F"-"H"| "K"| "FF"| "GG"| "HH"| "KK"|
      "X"-"Z"| "XX"| "YY"| "ZZ"| "f"-"h"| "k"-"n";
meta_property --> "P"| "P1"| "P2"| "Q"| "Q1"| "Q2";

dummy --> " ";
```

Operator definition

```
operator
      arrow_op; "--->"; ":"; "/"; "="; bind_op; "∧"; "∨"; "⇒";
predicate
      meta_arrow_symbol, meta_formula, arrow_function, arrow_function2.
```

9.2 Derivation system of category theory

Axioms

$1 : obj$

$0 : obj$

Inference rules

$$\frac{W1 \quad W2}{W1 \wedge W2}\ (\wedge \mathtt{I}) \qquad \frac{W1 \wedge W2}{W1}\ (\wedge \mathtt{E-left})$$

$$\frac{W1 \quad W1 \Rightarrow W2}{W2}\ (\Rightarrow \mathtt{E}) \qquad \frac{\begin{array}{c}[W1]\\ \vdots \\ W2\end{array}}{W1 \Rightarrow W2}\ (\Rightarrow \mathtt{I})$$

$$\frac{F : X \text{-->} Y}{dom(F) = X}\ (\mathtt{dom-I}) \qquad \frac{F : X \text{-->} Y}{cod(F) = Y}\ (\mathtt{cod-I})$$

$$\frac{dom(F) = X \quad cod(F) = Y}{F : X \text{-->} Y}\ (\mathtt{-->I})$$

$$\frac{F}{cod(F) : obj}\ (\mathtt{cod:obj-I}) \qquad \frac{F}{dom(F) : obj}\ (\mathtt{dom:obj-I})$$

$$\frac{F : obj}{cod(F) = F}\ (\mathtt{cod(obj)}) \qquad \frac{F : obj}{dom(F) = F}\ (\mathtt{dom(obj)})$$

$$\frac{cod(F) = dom(G)}{F.G}\ (\texttt{dot}-\texttt{I}) \qquad \frac{F.G}{cod(F) = dom(G)}\ (\texttt{dot}-\texttt{E})$$

$$\frac{F.G \quad G.H}{(F.G).H}\ (\texttt{dot}+\texttt{dot})$$

$$\frac{W(F) \quad F = G}{W(G/F)}\ (=\texttt{repl}-\texttt{formula})$$

$$\frac{F = G \quad H.F}{H.F = H.G}\ (\texttt{dot}-\texttt{left}) \qquad \frac{F = G \quad F.H}{F.H = G.H}\ (\texttt{dot}-\texttt{right})$$

$$\frac{F}{dom(F).F = F}\ (\texttt{left}-\texttt{id}) \qquad \frac{F}{F.cod(F) = F}\ (\texttt{right}-\texttt{id})$$

$$\frac{\begin{array}{c}[F.G = H.G]\\ \vdots\\ F = H \qquad G\end{array}}{G : monic}\ (\texttt{monic}-\texttt{I}) \qquad \frac{F : monic \quad G.F = H.F}{G = H}\ (\texttt{monic}-\texttt{E})$$

$$\frac{\begin{array}{c}[F.G = F.H]\\ \vdots\\ G = H \qquad F\end{array}}{F : epic}\ (\texttt{epic}-\texttt{I}) \qquad \frac{F : epic \quad F.G = F.H}{G = H}\ (\texttt{epic}-\texttt{E})$$

$$\frac{F.G : obj \quad G.F : obj}{F : iso}\ (\texttt{iso}-\texttt{I})$$

$$\frac{F : iso}{inv(F).F : obj}\ (\texttt{left}-\texttt{inv}) \qquad \frac{F : iso}{F.inv(F) : obj}\ (\texttt{right}-\texttt{inv})$$

$$\frac{X : obj}{0(X) : 0 \dashrightarrow X}\ (0-\texttt{map}) \qquad \frac{F : 0 \dashrightarrow X \quad G : 0 \dashrightarrow X}{F = G}\ (\exists!-0)$$

$$\frac{X : obj}{1(X) : X \dashrightarrow 0}\ (1-\texttt{map}) \qquad \frac{F : X \dashrightarrow 1 \quad G : X \dashrightarrow 1}{F = G}\ (\exists!-1)$$

$$\frac{X : obj \quad Y : obj}{pr1(X,Y) : X\mathbf{x}Y \dashrightarrow X}\ (\texttt{pr1}-\texttt{I}) \qquad \frac{X : obj \quad Y : obj}{pr2(X,Y) : X\mathbf{x}Y \dashrightarrow Y}\ (\texttt{pr2}-\texttt{I})$$

$$\frac{F : Z \dashrightarrow X \quad G : Z \dashrightarrow Y}{F * G : Z \dashrightarrow X\mathbf{x}Y}\ (\texttt{prod}-\texttt{I})$$

$$\frac{H.pr1(X,Y) = F \quad H.pr2(X,Y) = G}{H = F * G}\ (\exists!-\texttt{prod})$$

$$\frac{W(F)}{F}\ (\exists \mathtt{F})$$

Rewriting rules

$$\frac{W1 \wedge W2}{W2 \wedge W1}\ (\wedge \mathtt{com})$$

$$\frac{F = G}{G = F}\ (= \mathtt{sym})$$

$$\frac{(F.G).H}{F.(G.H)}\ (\mathtt{dot} - \mathtt{assoc})$$

$$\frac{(F * G).pr1(cod(F), cod(G))}{F}\ (\mathtt{pr1} - \mathtt{E}) \qquad \frac{(F * G).pr2(cod(F), cod(G))}{G}\ (\mathtt{pr2} - \mathtt{E})$$

9.3 Derived rules

The results proved in EUODHILOS are saved as either derived rules or theorems. They will be used to get other results later. In our experience, derived rules are often easier to use than theorems. The followings are some of the rather simple results saved as derived rules.

```
      6
[W1∧W2]
·(∧com{6})
W2∧W1
·(∧E-left{6})
W2
```

$\wedge E - right$

```
      7
[F=G]
·(=sym{7})
G=F
—(∃F{7})
 G
```

$\exists F(= right)$

```
   8        9
[W]   [W⇒W1]
—(⇒E{8, 9})          10
   W1          [W1⇒W2]
   ———(⇒E{8, 9, 10})
        W2
      ·(⇒I{9, 10})
      W⇒W2
```

$\Rightarrow\Rightarrow$

```
                    3
                [W1∧W2]
          11   ·(∧E-left{3})        3
[W1⇒(W2⇒W)]    W1              [W1∧W2]
———————————————(⇒E{3, 11})    ·(∧E-right{3})
    W2⇒W                        W2
    ———————————————————————————(⇒E{3, 3, 11})
                    W
                  ——(⇒I{11})
                W1∧W2⇒W
```

$\Rightarrow\Rightarrow to\wedge\Rightarrow$

```
   12       13
[F=G]   [G=GG]
(=repl-formula{12,13})
F=GG
```

$= trans$

```
 14
[F]
(left-id{14})    14
dom(F).F=F       [F]
——(=sym{14})    (left-id{14})
F=dom(F).F       dom(F).F=F
————————(=repl-formula{14})
            F=F
```

$F = F$

9.4 Proof examples

(A) An object is an isomorphism.

In the formulation, an object is the identity arrow of itself. Therefore we can say an object is an isomorphism.

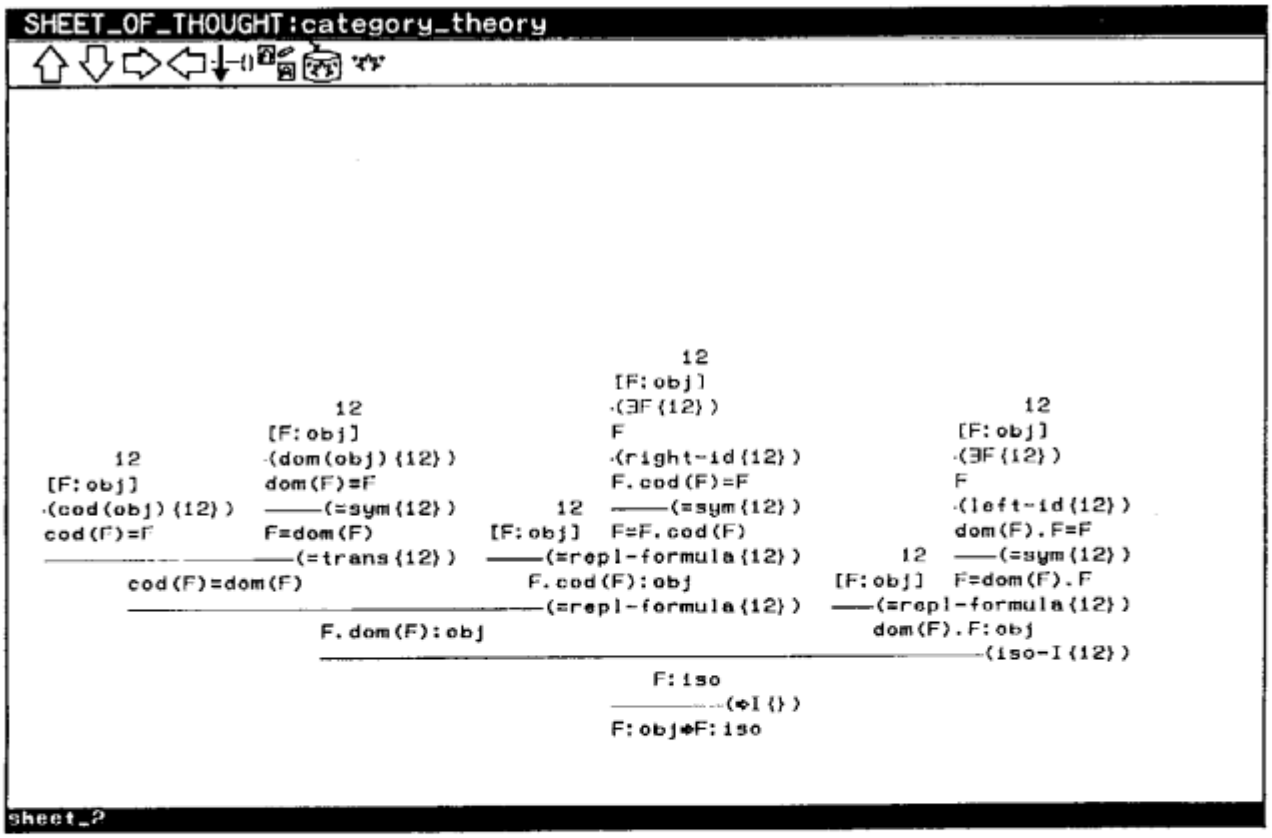


Figure 10: A proof of "$F : obj \Rightarrow F : iso$"

(B) The co-domain of a composition F.G of two arrows is equal to the co-domain of the constituent arrow G.

This proof example in the upper window uses a theorem illustrated in the middle window.

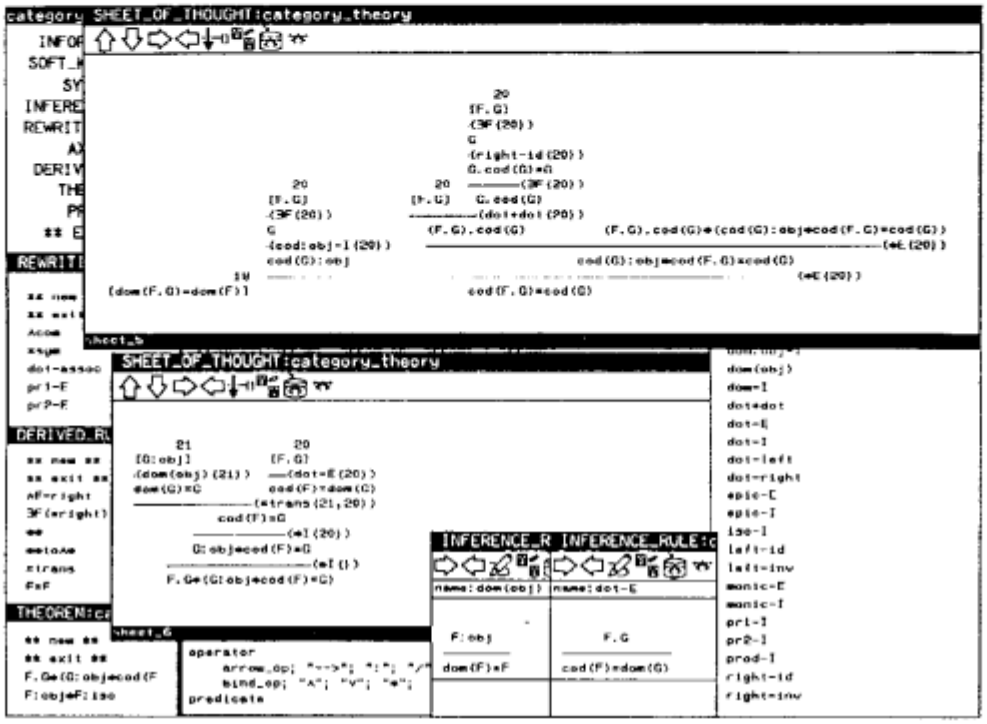


Figure 11: A proof of "$cod(F.G) = cod(G)$"

10 Miscellany

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

10.1 Smullyan's logical puzzles

$$\vdash \ (A * M) \bullet (A * M) = A \bullet ((A * M) \bullet (A * M))$$

In the Smullyan's book [Smullyan 85], variables such as "A", "B", "M" represent birds in an enchanted forest. Any bird in this forest will respond a bird's name if you call out a name of a bird. The expression "$A \bullet B$" represents the name when you call out the name of B to bird A. Mockingbird, represented by "M", is a bird characterized with the following property:

$$M \bullet x = x \bullet x \qquad \text{for any bird } x \text{ in the forest.}$$

This is defined as the first axiom named "Mocking bird" in Figure 12.

The collection of the birds inhabited in the forest also has the following property: For any birds A and B, there is a bird C such that

$$C \bullet x = A \bullet (B \bullet x) \qquad \text{for any bird } x \text{ in the forest.}$$

The bird C is represented by "$A * B$" and this property is also defined as an axiom with the name "Star."

A bird A is told being fond of the bird B if you call out B to A, then A calls the same name B back to you; i.e. $A \bullet B = B$.

The theorem indicates that every bird A of the forest is fond of at least one bird; i.e. $(A * M) \bullet (A * M)$.

10.2 Propositional modal logic (T)

$$\vdash \Diamond p \wedge \Box(p \supset q) \supset \Diamond(p \wedge q)$$

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

See Figure 13 for the screen layout of the proof.

10.3 Second-order reasoning

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

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

See Figure 14 for the screen layout of the proof.

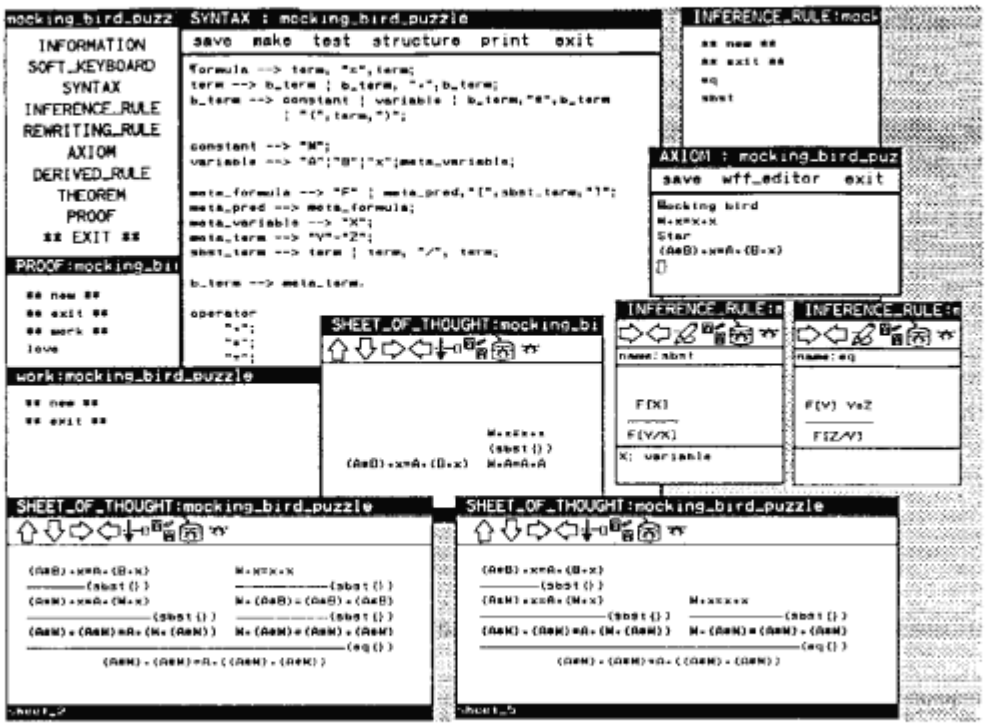


Figure 12: Smullyan's logical puzzles

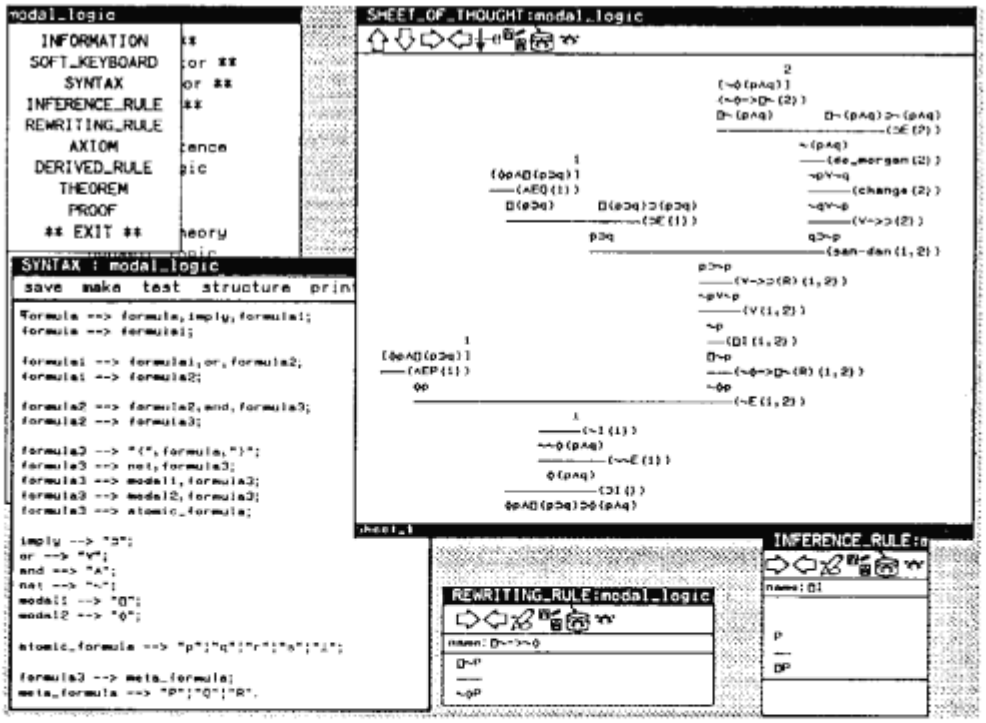
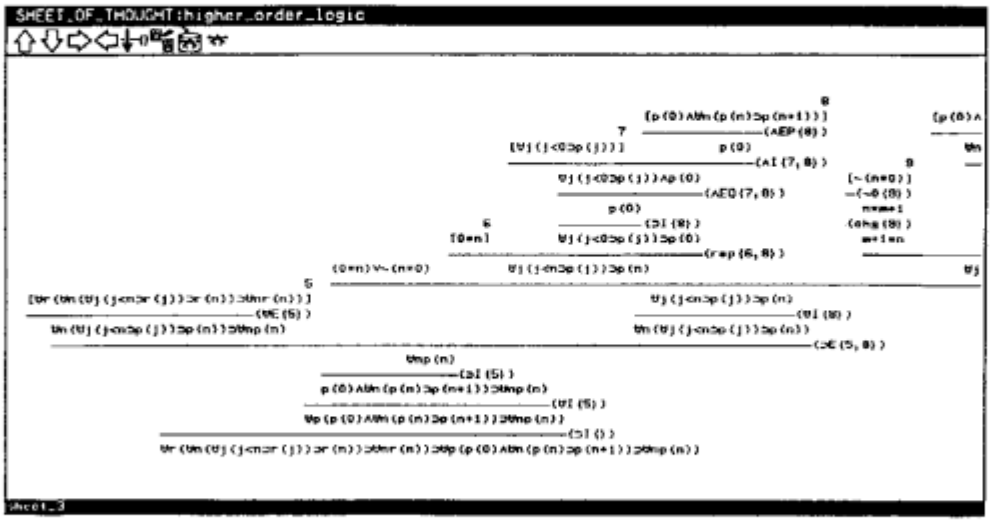


Figure 13: Propositional modal logic (T) and modal reasoning about programs



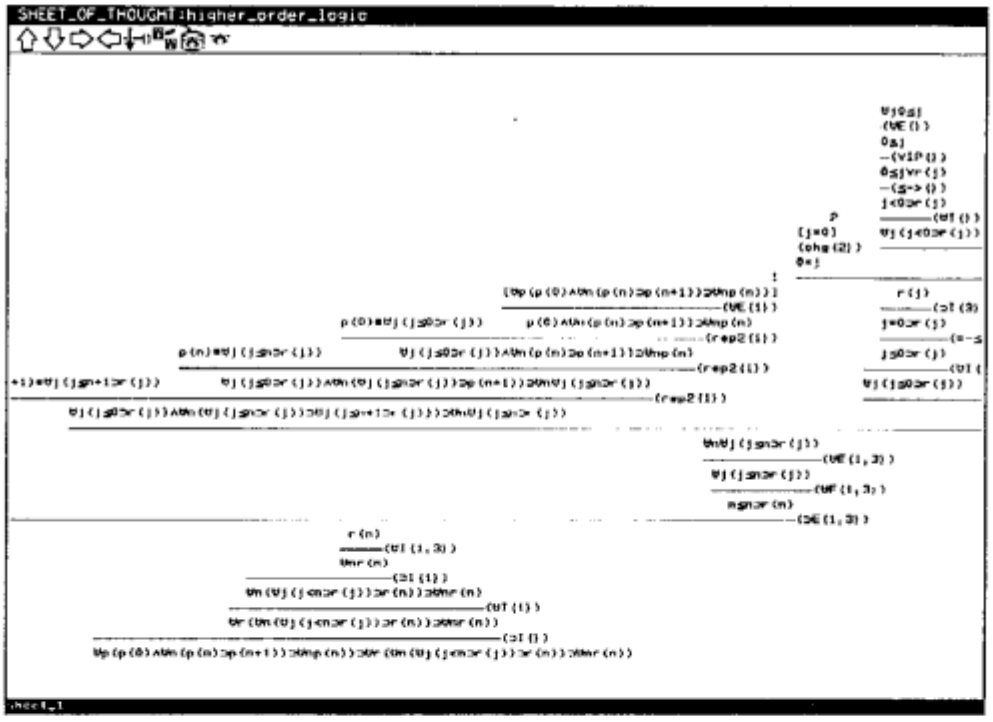


Figure 14: Simple equivalence proof by second-order reasoning

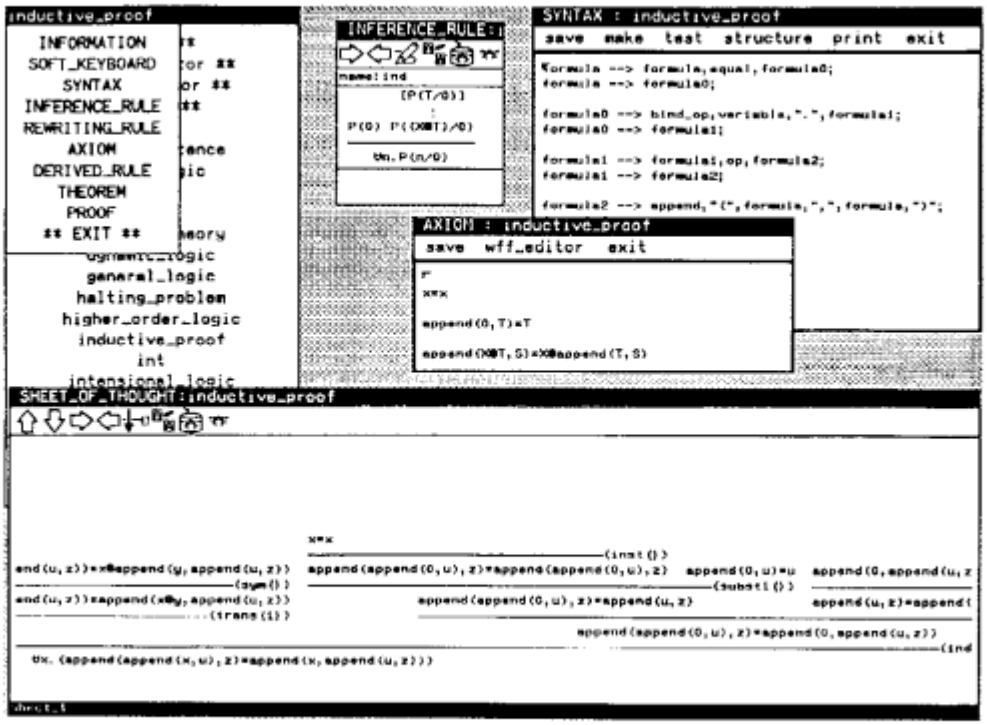


Figure 15: Proof by structural induction on list

10.4 Proof by structural induction on list

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

(associativity of append function)

See Figure 15 for the screen layout of the proof using the following structural induction on list:

$$\frac{\begin{array}{cc} & [P(T)] \\ & \vdots \\ P(0) & P(X.T) \end{array}}{\forall N.P(N)}$$

Acknowledgements

The authors would like to thank Dr. R. K. Meyer, Dr. J. K. Slaney, Prof. M. A. McRobbie, Dr. G. Meglicki and Mr. J. Riche(Australian National University) for their discussions on general logics and general reasoning systems. Special thanks are due to Dr. R. K. Meyer who kindly taught us his nice tag calculus. 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

[Burkholder 87] Burkholder, L. : The halting problem, SIGACT NEWS, Vol. 18, No. 3, pp. 48-60, 1987.

[Gallin 75] Gallin, D.: Intensional and higher-order modal logic, with applications to Montague semantics, North-Holland, 1975.

[Harel 84] Harel, D.: Dynamic logic, in Gabbay, D. and Guenthner, F. (eds.): Handbook of philosophical logic, Volume II : Extensions of classical logic, D. Reidel, pp. 497-604, 1984.

[Hoare 69] Hoare, C. A. R.: An axiomatic basis for computer programming, CACM, Vol. 12, No. 10, pp. 576-580, 583, 1969.

[Langer 25] Langer, S. K.: A set of postulates for the logical structure of music, Monist 39, pp. 561-570, 1925.

[Mac Lane 71] S. Mac Lane: Categories for the Working Mathematician, Springer-Verlag (1971).

[Martin-Löf 82] Martin-Löf, P.: Constructive mathematics and computer programming, pp. 153-179, in Cohen, L. J., Pfeiffer, H. and Podewski, K. P. (eds.), Logic, Methodology and Philosophy of Science VI, North-holland, 1982.

[Martin-Löf 84] Martin-Löf, P.: Intuitionistic type theory, Bibliopoplis, 1984.

[Meyer 90] Meyer, R. K.: Private communication, 1990.

[Minami 90] Minami, T., Sawamura, H., Satoh, K. and Tsuchiya, K.: EUODHILOS : A general-purpose reasoning assistant system - concept and implementation - , LNCS 383, Springer-Verlag, pp. 172-187, 1990.

[Minami 91] Minami, T., Yokota, K., Ohashi, K., Sawamura, H. and Ohtani, T.: A users manual for EUODHILOS, IIAS-RR-91, 1991 (to appear).

[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).

[Prawitz 65] Prawitz, D.: Natural deduction, Almqvist & Wiksell, 1965.

[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. Also appeared in ARP-TR-5/90, Australian National University, 1990.

[Sawamura 91a] Sawamura, H., Minami, T., Yokota, K. and Ohashi, K.: Potential of general-purpose reasoning assistant system EUODHILOS, to appear in Proc. of Software Science and Engineering, World Scientific Pub. Co., and IIAS-RR-91-8E, 1991.

[Sawamura 91b] Sawamura, H.: Specifying general logics and constructing proofs: A case study in EUODHILOS (in preparation), 1991.

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

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