General-Purpose Reasoning Assistant System

南, 俊朗
Graduate School of Information Science and Electrical Engineering, Kyushu University

https://doi.org/10.11501/3150887

出版情報：九州大学, 1998, 博士（理学）, 課程博士
バージョン：
権利関係：

7.1 Martin-Löf's Intuitionistic Type Theory

This example is a tiny subset of the intuitionistic type theory described in [2] and [55]. Based on the formulas-as-types notation, a type is represented as a formula that gives a formal specification and its element represents a program that satisfies the specification. This principal expression is represented in an intuitionistic type theory as a judgment of the form "x∈A", reads "x is a proof of a proposition A" in formulas-as-types interpretation, where "x" is an expression in λ-calculus and "A" is a first-order formula interpreted as a type. The judgment is naturally and well described in the EUODHILOS framework[78; 98; 100].

We present two description examples; one on EUODHILOS-I in the DCGo notation and the other on EUODHILOS-II in the BNF-based notation.

Description and Proof Example in DCGo-based framework

The language definition consists of four parts: an object language, a metalanguage, interface between the meta and object languages and the constructor declaration as follows. This is a typical classification of the descriptions in the DCGo notation.

Language:

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

% Object_language
judgement --> term, "∈", type;
term --> bind_op, variable, ".", term |
         term, ".", term |
         "(", term, ")" |
         "~", term |
         "inl", "(", term, ")" | "inr", "(", term, ")" |
         variable | constant |
         meta_term1, "(", term, ")" | meta_term;
```



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

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

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

% Constructor declaration
with_priority
 "~"; "v":left; "⊃":left; ".":left; "λ"; "∈";

without_priority
"inl", "inr", meta_term1.
```

It is notable that the syntax definition for the metalanguage is provided for defining inference rules schematically, and the operators have precedence in the indicated order as well as their associativity, and the functors or predicates, e.g., "inl" in the term "inl(x)", are listed simply by themselves or the nonterminals by which they are denoted, under the heading "predicate". As has been described in Chapter 4, the constructor declaration tells the parser and the unparser that the terminal name declared to be an operator or the terminal name denoted by the nonterminal character string is entitled to become the principal operator of the internal structure for an expression generated by the grammar rules.

Inference Rule:

The intuitionistic type theory is defined by a number of natural deduction style inference rules. For the purpose of illustration we consider just four rules and one

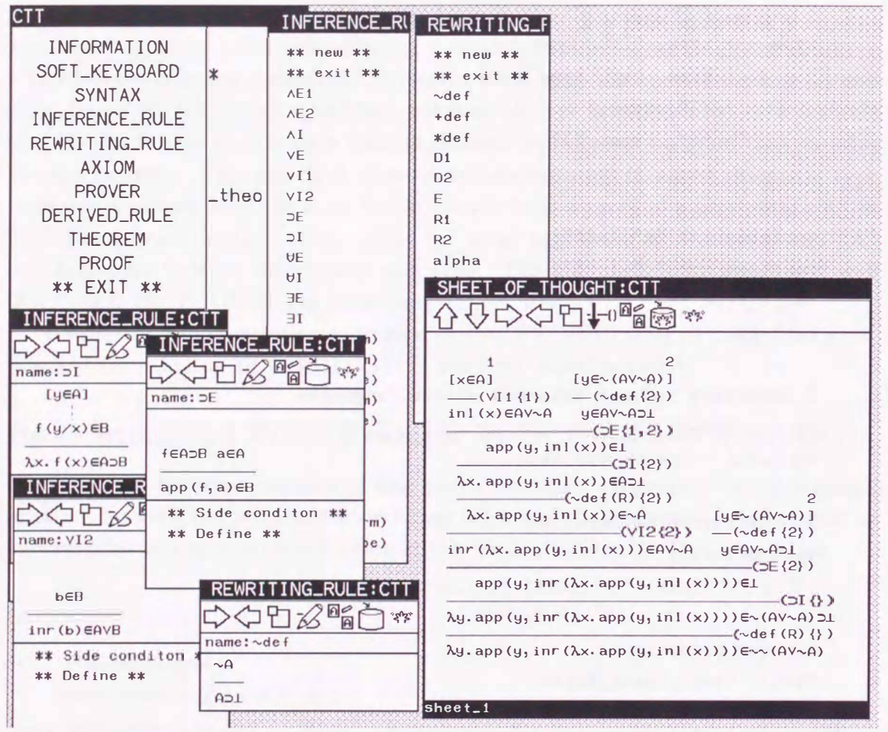


Figure 7.1: Intuitionistic Type Theory and Constructive Proof on EUODHILOS-I

rewriting rule. These are the rules for function introduction and elimination, the two rules for ∨-introduction, and the rewriting rule for the definition $\sim A = A \supset \bot$. As can be recognized in these example inference rule definitions, if we see only the right sides of the judgments the rule is exactly that of the natural deduction rules for the classical propositional logic.

⊃-I:

$$\frac{\begin{array}{c}[y \in A] \\ \vdots \\ f(y/x) \in B\end{array}}{\lambda x.f(x) \in A \supset B}$$

Side Condition: y is not free in B

⊃-E:

$$\frac{f \in A \supset B \quad a \in A}{app(f,a) \in B}$$

inl-I:

$$\frac{a \in A}{inl(a) \in A \vee B}$$

inr-I:

$$\frac{b \in B}{inr(b) \in A \vee B}$$

Rewriting Rule:

def:

$$\frac{A \supset \bot}{\sim A}$$

Proof Example:

Figure 7.1 displays the proof of the theorem $\sim\sim (A \vee \sim A)$. The theorem means that the law of the double negation of the excluded middle cannot be refuted. This is an instance of Glivenko's theorem that if A is any tautology of the classical propositional calculus then the proposition $\sim\sim A$ is always constructively valid. For further details about how the proof has been constructed using our various proof facilities and methods, refer to [102].

Description and Proof Example in EUODHILOS-II

Also in the description framework of EUODHILOS-II, the intuitionistic type theory can be formulated in a similar way. We present the actual description example for the same example logical system in order to provide a comparison example between the two EUODHILOS frameworks.

Language:

The syntax definition is as follows:

```
%ROOT Judgement
%META_VARIABLES
 Meta_Var = "[u-z][0-9]*" ;
 Meta_Term = "[a-kr-t][0-9]*" ;
 Meta_Type = "[A-GO-T][0-9]*" ;
 Meta_Judgement = "[IJ][0-9]*" ;
%PRODUCTIONS
 Judgement ::= Meta_Judgement ;
 Judgement ::= Meta_Judgement "(" Type ")" ;
 Judgement ::= Meta_Judgement "(" Term ")" ;
 Judgement ::= Term "∈" Type ;
 Judgement ::= Type "=" Type ;
 Judgement ::= Term "=" Term "∈" Type ;
 Op1 ::= "inl";
 Op1 ::= "inr";
 Op1 ::= "car";
 Op1 ::= "cdr";
 Op2 ::= "ap" ;
 Op2 ::= "cons" ;
 Op2 ::= "E" ;
 Op3 ::= "D" ;
 Variable ::= Meta_Var ;
 Term ::= Variable ;
 Term ::= Meta_Term ;
 Term ::= Meta_Term "(" List_Of_Term ")" ;
 Term ::= Op1 "(" Term ")" ;
 Term ::= Op2 "(" Arg2 ")" ;
 Term ::= Op3 "(" Arg3 ")" ;
 Term ::= Term "/" Term ;
 Term ::= "λ" @Term "." [ Term ] ;
 Arg2 ::= Term "," Term ;
 Arg3 ::= Term "," Term "," Term ;
 List_Of_Term ::= Term ;
 List_Of_Term ::= Term "," List_Of_Term ;
 Type ::= "⊥" ;
 Type ::= Meta_Type ;
 Type ::= Meta_Type "(" List_Of_Term ")" ;
 Type ::= "(" Type ")" ;
```



```
 Type ::= "¬" Type ;
 Type ::= Type "/" Type ;
 Type ::= Type "∧" Type ;
 Type ::= Type "∨" Type ;
 Type ::= Type "⊃" Type ;
 Type ::=
  "(" "∀" @Variable "∈" Type ")" [ Type ] ;
 Type ::=
  "(" "∃" @Variable "∈" Type ")" [ Type ] ;
```

Axiom:

There are two axioms, $a = a \in A$ and $A = A$, which give the identity properties for the elements and the types, respectively.

Inference Rule:

⊃I:

$$\frac{\begin{array}{c}[x \in A] \\ \vdots \\ f \in B\end{array}}{\lambda x.f \in A \supset B}$$

Side Condition: `(NOT-FREE ("x" . Variable) ("B" . Type))`

Note that the rule body looks the same to the previous example, whereas the side condition is specified in a different style of description. The other rules are omitted here.

Rewriting Rule:

¬def:

$$\frac{A \supset \bot}{\neg A}$$

cons:

$$\frac{cons(car(t), cdr(t))}{t}$$

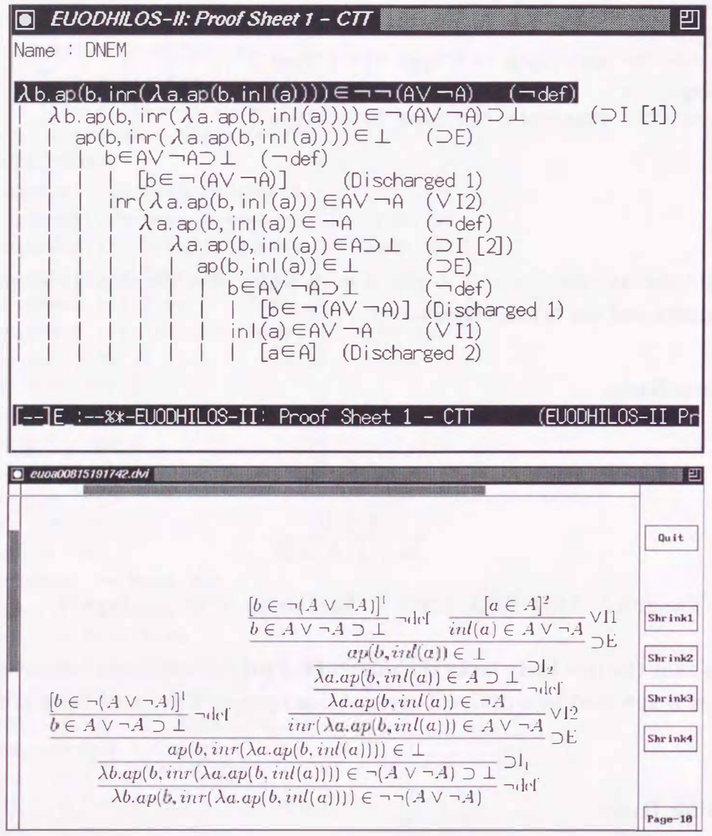


Figure 7.2: A Proof of the Double Negation of the Excluded Middle Law in Intuitionistic Type Theory

Proof Example:

The screen image in Figure 7.2 shows the proof of the double negation of the excluded middle law. In the upper window, the proof is displayed in a full-tree representation, whereas in the lower window the same proof is displayed with a DVI previewer by using the LaTeX macros for having a better view of the proof trees.

7.2 Modal Logic

Modal logic[37] is a variety of classical logic extended by adding two modal expressions: $\Box A$ and $\Diamond A$. These propositions assert that "it is *necessary* that proposition A holds" and "it is *possible* that proposition A holds," respectively.

In this section we present two formulation examples. The first example is a standard one of propositional modal logic and the other one is a logic for program verification which is formulated by using modal operators, where for each program p, two modal operators $[p]$ and $< p >$ are introduced.

(A) Modal Logic T

Language:

Two modal operators, $\Box$ (necessity) and $\Diamond$ (possibility), are added to the language of classical propositional logic:

```
%ROOT Formula
%META_VARIABLES
 Identifier = "[A-Z][A-Z0-9]*" ;
%PRODUCTIONS
 AtomicFormula ::= "⊥" ;
 AtomicFormula ::= Identifier ;
 Formula ::= AtomicFormula ;
 Formula ::= "¬" Formula ;
 Formula ::= "◇" Formula ;
 Formula ::= "□" Formula ;
 Formula ::= "(" Formula ")" ;
 Formula ::= Formula "∧" Formula ;
 Formula ::= Formula "∨" Formula ;
 Formula ::= Formula "⊃" Formula ;
 Formula ::= Formula "≡" Formula ;
```

Axiom:

The first three axioms come from ordinary propositional logic, and the rest are specific to the modal logic T.

(1) $A \supset (B \supset A)$

(2) $(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$

(3) $(\neg A \supset \neg B) \supset (B \supset A)$

(4) $\Box A \supset A$

(5) $\Box(A \supset B) \supset (\Box A \supset \Box B)$

Inference Rule:

$$\frac{A \quad A \supset B}{B}\ MP \qquad \frac{A}{\Box A}\ \Box$$

Rewriting Rule:

We will use the following rewriting rules:

$$\frac{A \supset \bot}{\neg A}\ \neg def \qquad \frac{\neg\neg A}{A}\ \neg\neg \qquad \frac{\neg\Box\neg A}{\Diamond A}\ \Diamond def$$

$$\frac{\neg\Diamond\neg A}{\Box A}\ \Box def \qquad \frac{(A \supset B) \wedge (B \supset A)}{A \equiv B}\ \equiv def$$

$$\frac{\neg(\neg A \vee \neg B)}{A \wedge B}\ \wedge def \qquad \frac{A \wedge B}{B \wedge A}\ \wedge exchange$$

$$\frac{\neg A \supset B}{A \vee B}\ \vee def \qquad \frac{A \vee B}{B \vee A}\ \vee exchange$$

$$\frac{\neg(A \wedge B)}{\neg A \vee \neg B}\ de\_morgan1 \qquad \frac{\neg(A \vee B)}{\neg A \wedge \neg B}\ de\_morgan2$$

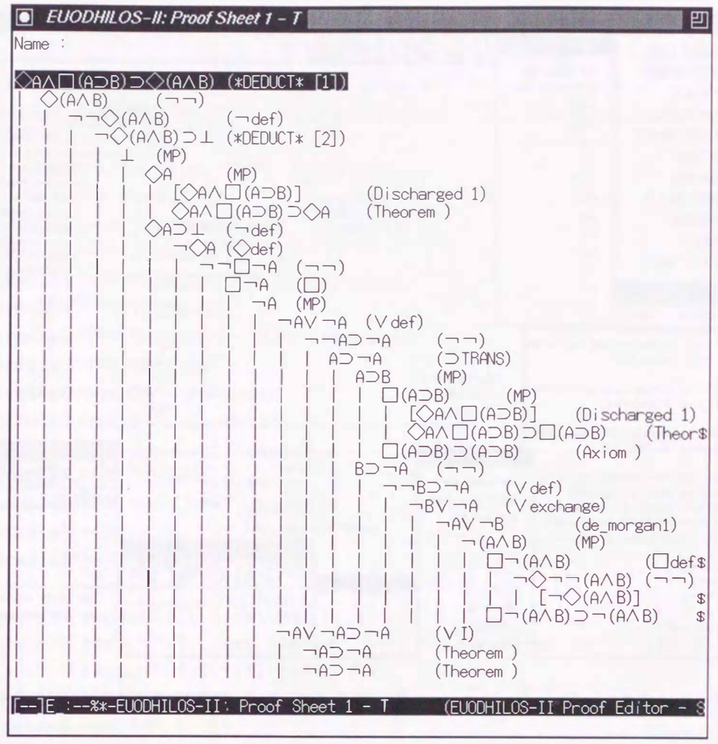


Figure 7.3: A Proof of $\Diamond A \wedge \Box(A \supset B) \supset \Diamond(A \wedge B)$ in Modal Logic T

Proof Example:

Figure 7.3 shows a proof of $\Diamond A \wedge \Box(A \supset B) \supset \Diamond(A \wedge B)$. To prove this proposition, we use propositional theorems. Without these theorems, the proof would be much larger and more difficult.

Proof Example in the DCGo Framework:

Figure 7.4 is an example in DCGo framework, where a proof example of "a strong correctness assertion is implied from a termination assertion and a weak correctness assertion" is displayed in the screen.

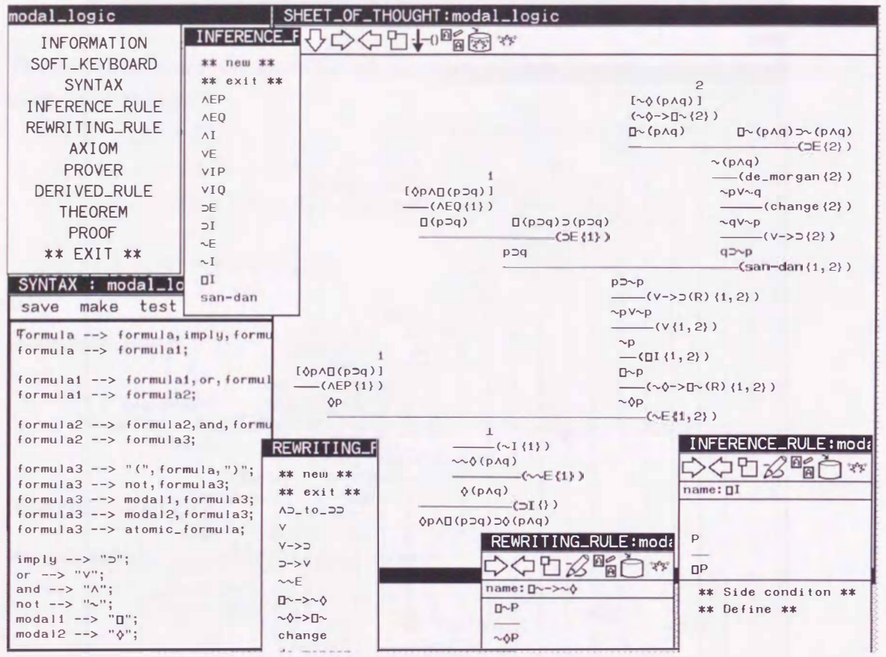


Figure 7.4: A Proof of $<>p \wedge [](p \supset q) \supset <>(p \wedge q)$ on EUODHILOS-I

Theorem:

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

(B) Dynamic Logic

Dynamic logic[33] is a kind of multi-modal logic which is an extension to classical logic. The principal formulas in dynamic logic are the dynamic formulas of the form "[a]p" and its 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 DCGo framework of EUODHILOS-I.

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, ")";
variable --> "x" | "y" | "z" | "n";
constant --> "0" | "1";

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;

regular_program2 --> meta_program;
formula3 --> meta_formula, "(", term, ")";
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".

with_priority
  "!"; "x"; ("+", "-"); ("<", "["); ("~", ">", "≥", "=");
  "∧"; "∨"; "⊃"; "≡"; ("?", ":="); "*"; (";", "|");
without_priority
  meta_formula.
```

Axiom:

(1) $[Q?]P \equiv (Q \supset P)$ (test)
(2) $[X := T]P(X) \equiv P(T)$ (assignment axiom)
(3) $[A; B]P \equiv [A][B]P$ (composition)
(4) $< A; B > P \equiv < A >< B > P$ (composition)
(5) $[A|B]P \equiv ([A]P \wedge [B]P)$ (nondeterministic selection)
(6) $P(X) \wedge X = T \supset P(T)$ (substitution)
(7) $x = 0 \supset (x = 0 \supset true)$ (arith)

Inference Rule:

modus ponens:

$$\frac{P \quad P \supset Q}{Q}$$

necessitation:

$$\frac{P \supset Q}{[A]P \supset [A]Q}$$

invariance:

$$\frac{P \supset [A]P}{P \supset [A*]P}$$

convergence:

$$\frac{n \geq 0 \wedge P(n+1) \supset < A > P(n)}{n \geq 0 \wedge P(n) \supset < A* > P(0)}$$

composition 1:

$$\frac{P \supset < A > Q \quad Q \supset < B > R}{P \supset < A; B > R}$$

composition 2:

$$\frac{P \supset [A]Q \quad Q \supset [B]R}{P \supset [A; B]R}$$

derived-rule1:

$$\frac{P \supset < A > Q \quad R \supset [A]S}{P \wedge R \supset < A > (Q \wedge R)}$$

replacement 1:

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

replacement 2:

$$\frac{P \supset Q \quad Q \equiv R}{P \supset R}$$

symmetricity:

$$\frac{P \equiv Q}{Q \equiv P}$$

Rewriting Rule:

def:

$$\frac{[A]P}{\sim < A > \sim P}$$

neg-elim:

$$\frac{\sim P \equiv \sim Q}{P \equiv Q}$$

double-neg-elim:

$$\frac{\sim\sim P}{P}$$

arithmetic:

$$\frac{n \geq 0 \wedge x = n}{x \geq 0}$$

true-elim:

$$\frac{true \wedge P}{P}$$

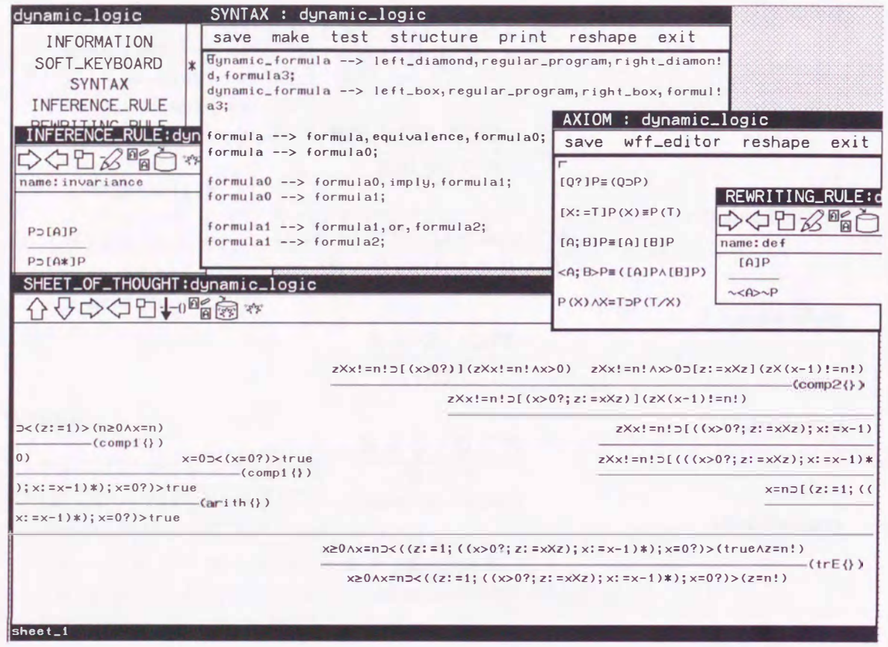


Figure 7.5: Total Correctness Theorem in Dynamic Logic

Lemma:

(1) $< (x = 0)? > true \equiv (x = 0 \supset true)$

(2) $n \geq 0 \wedge x = n + 1 \supset < (x > 0)? > (x = n + 1)$

(3) $x = n + 1 \supset < z := x \times z > (x = n + 1)$

(4) $x = n + 1 \supset < x := x - 1 > (x = n)$

(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) $z \times x! = n! \supset [(x = 0)?](z = n!)$

Proof Example:

One of the example proofs in this logic is the following properties of a factorial program:

(1) Termination:

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

(2) Partial Correctness:

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

(3) Total Correctness:

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

Figure 7.5 is an example screen of proving the total correctness theorem of the factorial program.

7.3 Intensional Logic

Intensional logic[25] 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 are all well described in the DCGo notation.

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(_), colon, type(_);
meta_type(_) --> "a" | "b" | "c" | "T" | "T1" | "T2" | "T3";
```

```
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((T1, T2)) --> bind_op, variable(T1), ".", term5(T2);
term4(T) --> term5(T);
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) --> "∨", 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(_), ")";
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), ")".

with_priority
```



```
    (":", ",");
    ("^", "∨", "[]", "<>");
    ("•", "~");
    "{";
    bind_op;
    "=";
    "∧";
    "∨";

  without_priority
    meta_term2, pred_const.
```

Axiom:

(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) $[](^{\vee}F:(s,a) =^{\vee} G:(s,a)) = (F:(s,a) = G:(s,a))$
(6) $^{\vee}\hat{}A:a = A:a$

Inference Rule:

Reflection-1:

$$\frac{beweis(A)}{A}$$

Reflection-2:

$$\frac{A}{beweis(A)}$$

=>I:

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A => B}$$

Replace-1:

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

Replace-2:

$$\frac{A(B) = A(R) \quad R = S}{A(B) = A(S)}$$

Symmetricity:

$$\frac{R = S}{S = R}$$

Rewriting Rule:

∀-Definition:

$$\frac{\lambda X : a.P : t = \lambda X : a.true : t}{\forall X : a.P : t}$$

Brace convention:

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

Notational convention:

$$\frac{F \bullet G}{F(G)}$$

Lemma:

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

Proof Example:

The following metatheorem is ingeniously proved using the idea of the reflection principle[114].

Generalization rule:

$$\vdash P : t \Rightarrow \vdash \forall x : a.P : t$$

Figure 7.6 illustrates a proof of this 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 possible world semantics. Under the defined intensional logic, the following complicated intensional formula:

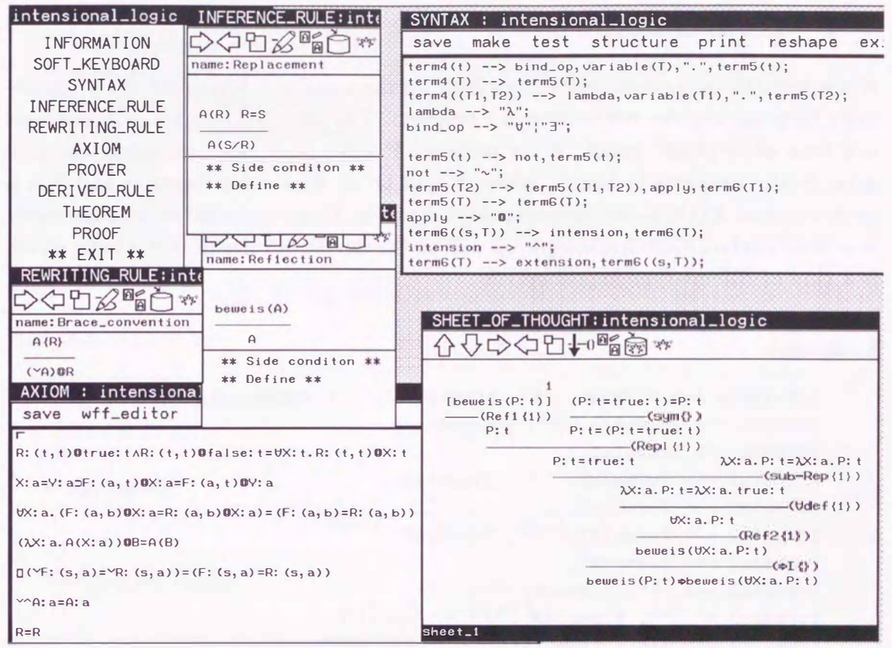


Figure 7.6: A Proof of the Generalization Rule in Intensional Logic

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

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

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

This theorem also has been proved easily.

7.4 Hoare Logic

Hoare logic[35] 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 the property P holds, then after executing the program S, the property 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 DCGo framework, as well as the inference rules of Hoare logic which is a kind of Hilbert-type logical system.

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;

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.

with_priority
 "!"; "*"; "+"; "="; "~"; "∧"; "∨"; "⊃"; ":="; "while"; "if"; "{".
```

Axiom:

Conjunction-elimination:

$$E \wedge F \supset E$$

Substitution:

$$P(X) \wedge X = T \supset P(T)$$

Assignment axiom:

$$P(T)\{X := T\}P(X)$$

Arithmetic:

$$true \supset 1 = 0!$$

Inference rule:

Consequence rule 1:

$$\frac{E \supset F \quad F\{A\}G}{E\{A\}G}$$

Consequence rule 2:

$$\frac{E\{A\}F \quad F \supset G}{E\{A\}G}$$

Composition rule:

$$\frac{E\{A\}F \quad F\{B\}G}{E\{A; B\}G}$$

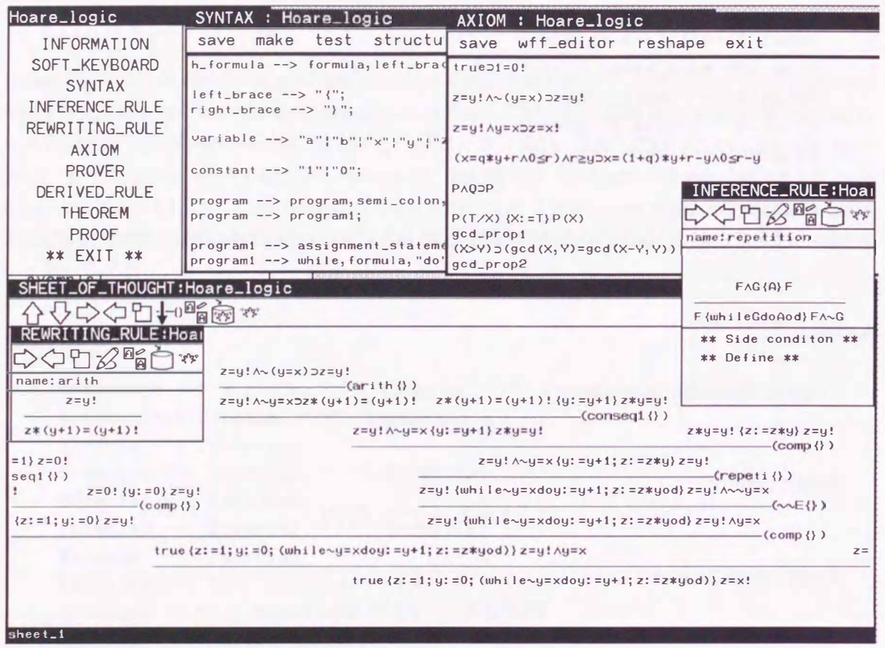


Figure 7.7: Partial Correctness Proof of a Factorial Program in Hoare Logic

Conditional rule:

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

Repetition rule:

$$\frac{F \wedge G\{A\}F}{F\{while\ G\ do\ A\ od\}F\wedge \sim G}$$

Rewriting rule:

Arithmetic rule:

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

Proof Example:

The screen layout of the proof of the following partial correctness assertion of a factorial program is shown in Figure 7.7:

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

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

7.5 Linear Logic

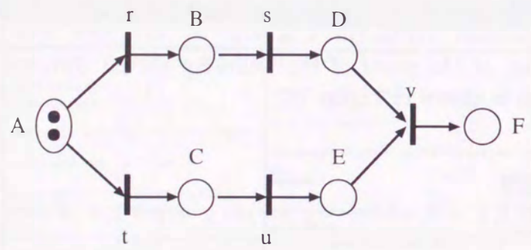
Linear logic was originated from J.-Y. Girard[26] and is usually described in the sequent style formalization. Unlike the classical logic LK, it lacks the two structural rules of *weakening* and *contraction*, so that it is suitable for handling the properties of finite resources. From these differences the free creation and deletion of an arbitrary proposition are prohibited in Linear Logic.

Petri net[54] is a computation model of concurrent processes, and is used to analyze their properties. The behavior of Petri net can be represented naturally in linear logic[31], because the tokens in a Petri net has the property of resources. To be precise, the diagram:

denotes that "if we have a token on condition A, then it is possible to fire event t. Firing t exhausts the token on A and provides a token on B." This can be translated into the linear logic sequent $A \vdash B$.

The reachability problem, which is common in Petri net theory, can be replaced with the provability problem in linear logic.

Consider the following Petri net:

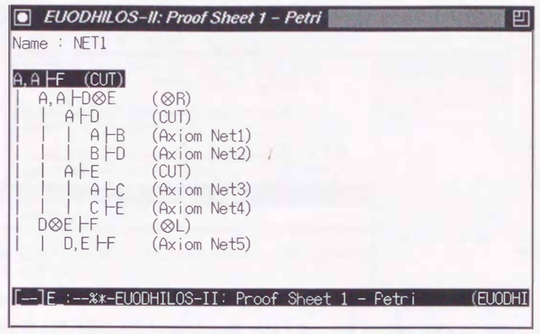


The problem is to determine whether the proposition "if there are two tokens on A, a token can reach F" holds. In linear logic, this problem is expressed as $A, A \vdash F$.

Language:

We specify a language system that can represent the above problem.

```
%ROOT Sequent
%META_VARIABLES
 Meta_Formula = "[P-R][0-9]*" ;
 Meta_Formula_List = "[L-N][0-9]*" ;
%PRODUCTIONS
 Sequent ::= Formula_List "⊢" Formula_List ;
 Sequent ::= Formula_List "⊢" ;
 Sequent ::= "⊢" Formula_List ;
 Formula_List ::= Meta_Formula_List ;
 Formula_List ::= Formula ;
 Formula_List ::= Formula_List "," Formula_List ;
 Atomic_Formula ::= "A" ;
 Atomic_Formula ::= "B" ;
 Atomic_Formula ::= "C" ;
 Atomic_Formula ::= "D" ;
 Atomic_Formula ::= "E" ;
 Atomic_Formula ::= "F" ;
 Formula ::= Meta_Formula ;
 Formula ::= Atomic_Formula ;
 Formula ::= "(" Formula ")" ;
 Formula ::= Formula "⊗" Formula ;
```

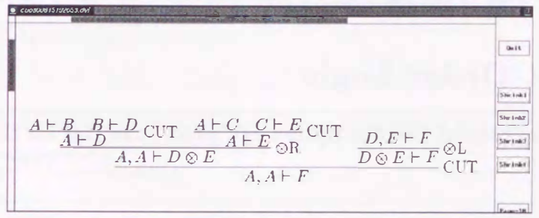


Figure 7.8: Proof of the Reachability Problem.

Axiom:

Transitions of the above Petri net are expressed as follows:

$$r: A \vdash B \qquad s: B \vdash D \qquad v: D, E \vdash F$$
$$t: A \vdash C \qquad u: C \vdash E$$

Inference Rule:

We will use only the following four inference rules in this example:

(1) CUT:

$$\frac{L \vdash P \quad P \vdash Q}{L \vdash Q}$$

(2) EXCHL:

$$\frac{P, Q \vdash M}{Q, P \vdash M}$$

(3) $\otimes$L:

$$\frac{P, Q \vdash M}{P \otimes Q \vdash M}$$

(4) $\otimes$R:

$$\frac{L1 \vdash P \quad L2 \vdash Q}{L1, L2 \vdash P \otimes Q}$$

Proof Example:

The formula to be proved is $A, A \vdash F$. Its simplest proof is shown in Figure 7.8, where the lower part is the same proof tree displayed by the DVI previewer.

7.6 First-Order Logic

In this section we present two description and proof examples for classical first-order logic.

(A) Unsolvability of the Halting Problem

Axiom:

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

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

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

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

where the meaning of each predicate is as follows:
A(x): x is an algorithm,
C(y): y is a computer program in some programming language,

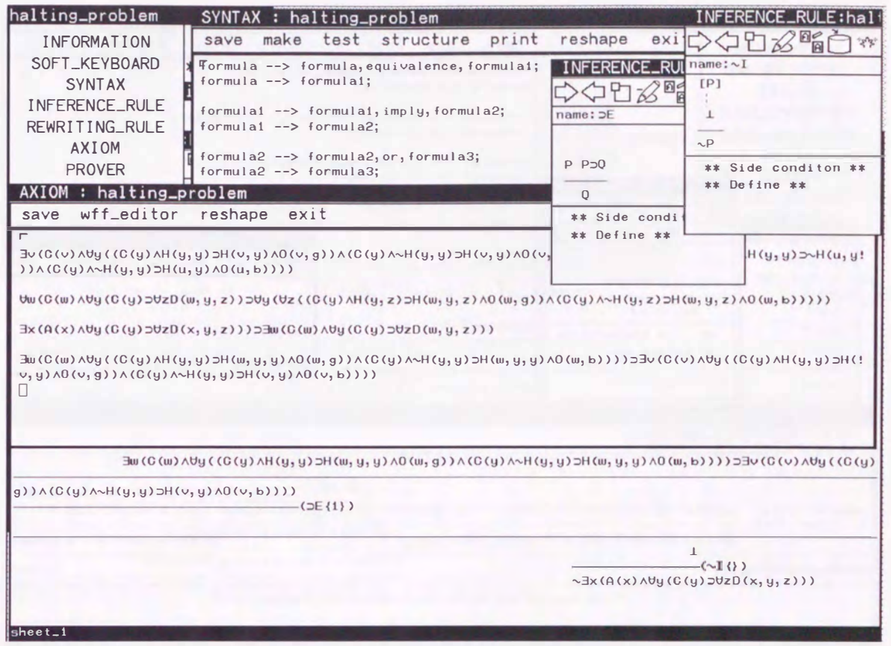


Figure 7.9: Unsolvability of the Halting Problem in First-Order Logic

D(x,y,z): x is able to decide whether y halts with given input z,
H(x,y): x halts with given input y,
H(x,y,z): x halts with given inputs y and z,
O(x,g): x outputs g; i.e. x halts, and
O(x,b): x outputs b; i.e. x does not halt.

Proof Example:

Figure 7.9 shows part of a proof of the theorem that no algorithm to solve the halting problem[10] exists:

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

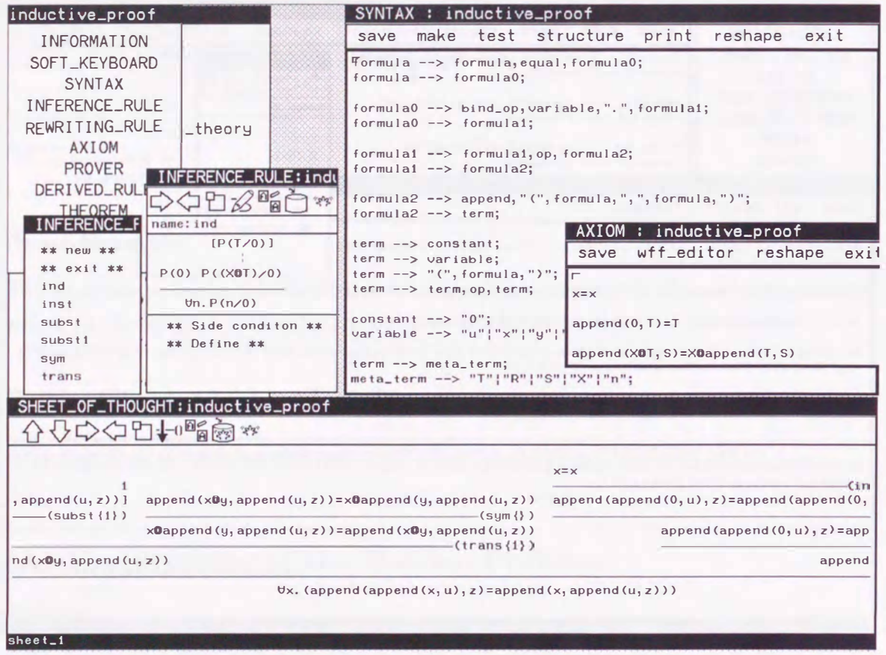


Figure 7.10: Structural Induction Proof in First-Order Logic

(B) Structural Induction on List Structure

As another example proof for first-order logic we take a recursive data structure which is popularly used for defining new data types.

Proof Example:

Figure 7.10 is a proof for the theorem:

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

which expresses the associativity of the append function.

The structural induction is a well-used method for proving useful properties for the data structures that are defined in terms of recursion.

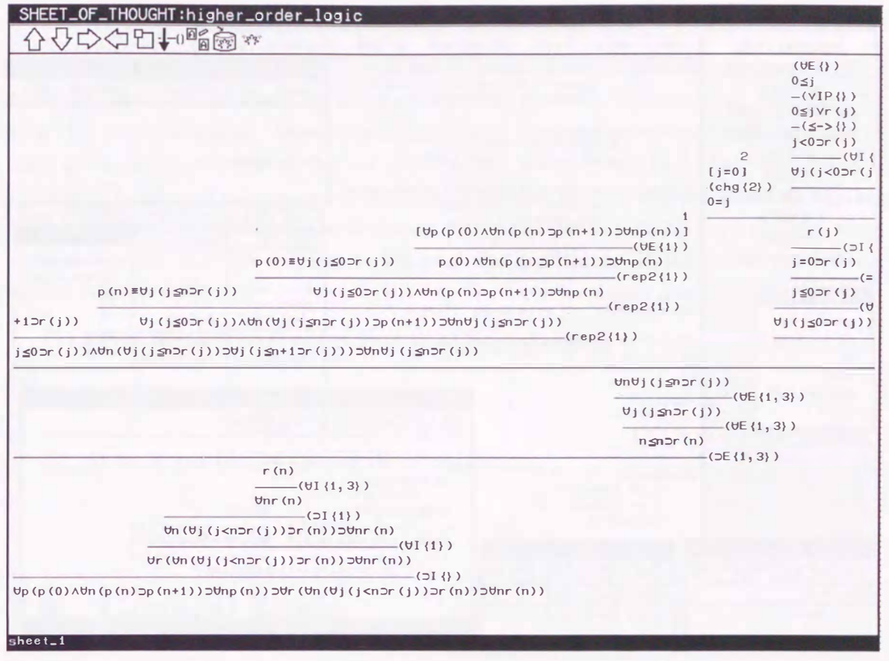


Figure 7.11: Proof Examples of Second-Order Logic

7.7 Second-Order Logic

We present a proof of second-order logic as an example of higher-order logic.

Proof Example:

Figure 7.11 proves that the principle of the mathematical induction is equivalent to the principle of the complete induction:

$$\vdash \forall p[p(0) \wedge \forall n(p(n) \supset p(n+1)) \supset \forall np(n)] \equiv \forall r[\forall n(\forall j(j < n \supset r(j)) \supset r(n)) \supset \forall nr(n)]$$

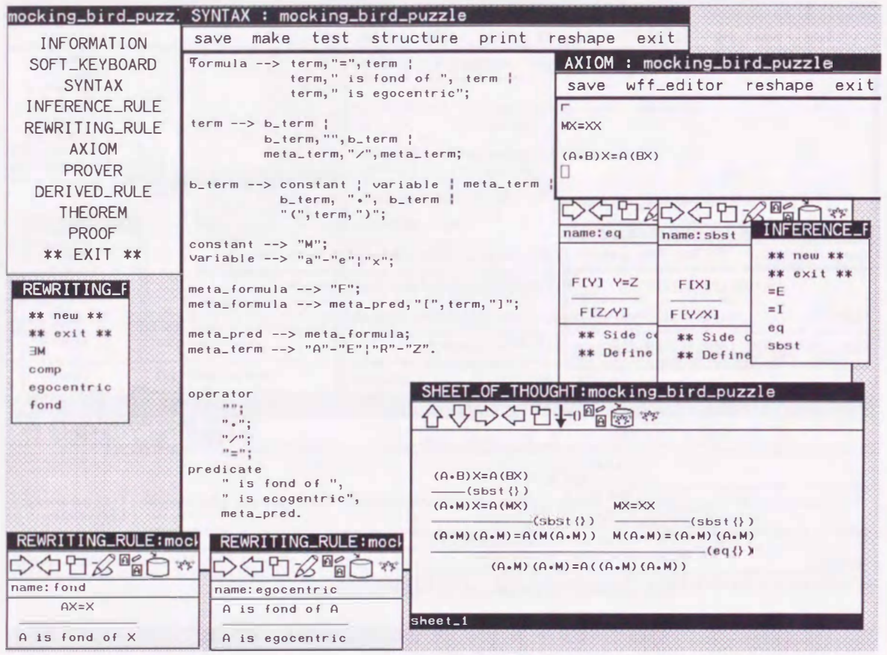


Figure 7.12: Combinatory Logic: Mocking Bird Puzzle

7.8 Combinatory Logic

This proof example is taken from the Smullyan's book[108] in which each combinator is interpreted as a bird in a wood, where each bird will reply with a bird name as it is called with a bird name. In this formulation the proposition "$z = x \bullet y$" says that the bird x will reply with the bird z as it is called with y.

Axiom:

(1) Mockingbird condition

$$\forall x.mx = xx$$

(2) Composition

$$\forall x \forall y \forall w.(x \bullet y)w = x(yw)$$

Proof Example:

The proof example in Figure 7.12 is the one for the first theorem below. In this example, the x in the theorem is represented by A and a metavariable was used for y in the above theorem. The metavariable for y is instantiated gradually and in the final proof of the window, it is $(A \bullet M)(A \bullet M)$, which is the bird that the bird A is fond of. This is an example proof of problem solving by starting the proof procedure with inputting a metavariable to the place where the solution should appear and when the proof terminates the solution to the problem is obtained as the substituted result to the metavariable.

(1) Every bird of the forest is fond of at least one bird.

$$\vdash \forall x \exists y(xy = y)$$

(2) At least one bird is egocentric or narcissistic.

$$\vdash \exists x(xx = x)$$

7.9 General Logic

General logic[105] is a kind of Gentzen-type formal system which yields a unified account of a fairly wide range of logical systems. Diverse logics are displayed as variations on a single theme. Such a general logic have been very successfully and smoothly handled on the DCGo framework by specifying those variations on a single theme as rewriting rules.

Proof Example:

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

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

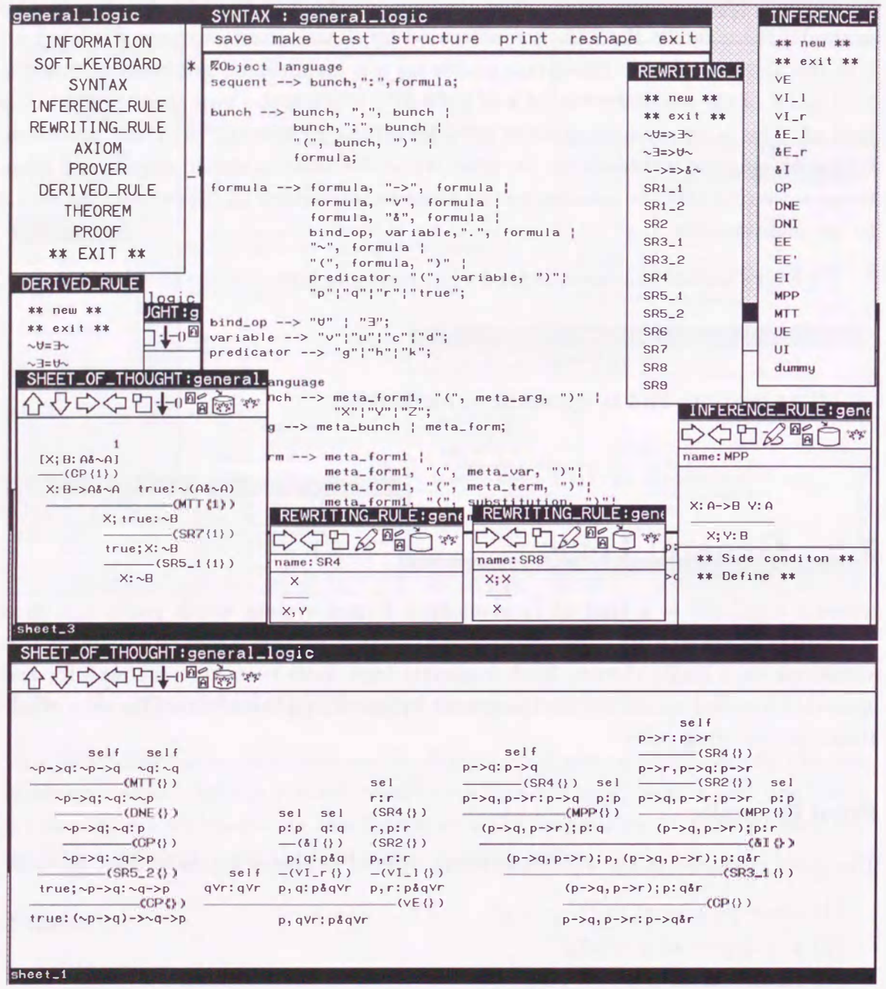


Figure 7.13: Proof Examples in General Logic

7.10 Relevant Logic

The relevant logic[58] we have taken is an implicational fragment of relevant logic, $R_{\rightarrow}$. Dependency for this logic is specified as a tag of a formula, differently from the usual set-theoretic dependency calculus, and then the tag is a composite formed from combinators satisfying some reduction rules. Tag of $R_{\rightarrow}$ is to stipulate dependency of an inference so as to yield a conclusion relevantly from an antecedent.

Inference Rule:

(1) Tag rule C: $C\alpha\beta\gamma = \alpha\gamma\beta$

$$\frac{(T_1 * T_3) * T_2 => P}{(T_1 * T_2) * T_3 => P}$$

(2) Tag rule B: $B\alpha\beta\gamma = \alpha(\beta\gamma)$

$$\frac{T_1 * T_2 * T_3 => P}{T_1 * (T_2 * T_3) => P}$$

(3) Tag rule W: $W\alpha\beta = \alpha\beta\beta$

$$\frac{T_1 * T_2 => P}{T_1 * T_2 * T_2 => P}$$

(4) →E:

$$\frac{T_1 => P \rightarrow Q \quad T_2 => P}{T_1 * T_2 => Q}$$

This is a universal method for dealing with the logics that have the different methods of dependency calculation mechanism from the one taken in EUODHILOS systems. Note that these rules can be specified by using the rewriting rules. For example the Tag rule C becomes like this:

(1') Tag rule C: $C\alpha\beta\gamma = \alpha\gamma\beta$

$$\frac{(T_1 * T_3) * T_2}{(T_1 * T_2) * T_3}$$

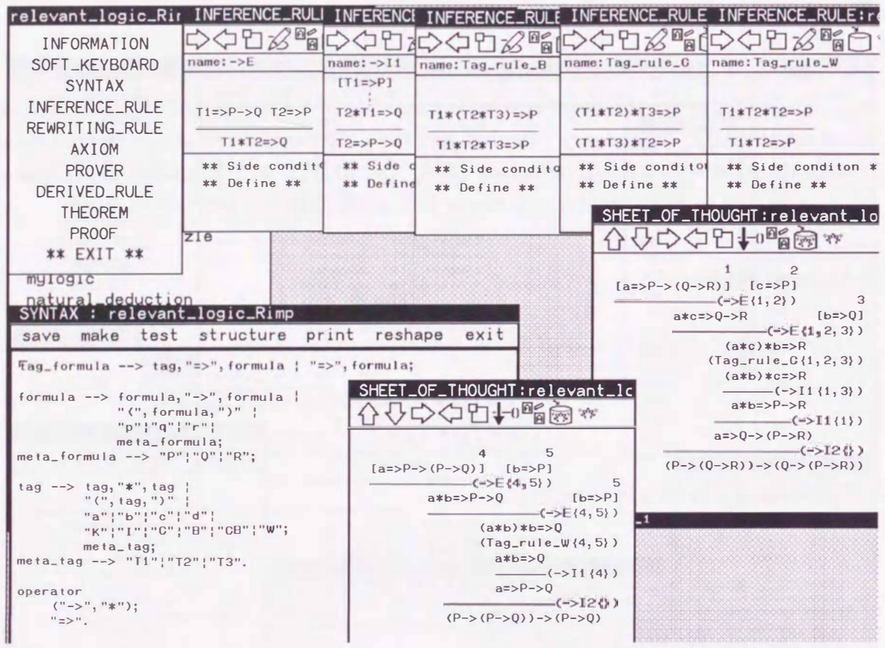


Figure 7.14: Proof Examples in Relevant Logic

Proof Example:

The following typical theorems of relevant logic has been proved in this formulation.

(1) (Permutation)

$$(P \to (Q \to R)) \to (Q \to (P \to R))$$

(2) (Prefixing)

$$(P \to Q) \to ((R \to P) \to (R \to Q))$$

(3) (Contraction)

$$(P \to (P \to Q)) \to (P \to Q)$$

(4) (Self-Implication)

$$P \to P$$

7.11 Category Theory

Category theory[52] was formulated as the theory of "naturalness." It is a mathematical theory of objects and arrows, or morphisms, which is used in various theories in computer science.

An elementary category theory have been built up. In this formulation we chose arrow-only notation; i.e. objects are identified with the identity arrow in order to make the theory easy to handle with. Note that this example theory deal with the partially-defined concept of composition of arrows. A composite expression $F.G$ of two arrows F and G does not always make sense. In this formulation the proposition given as an arrow expression means to state that such an arrow exists. For example, if we put $F.G$ as a proposition, it means that F and G are composable.

Proof Example:

Figure 7.15 is a sample proof. In the upper center part of the upper screen in the figure is a proof of the theorem:

$$cod(F.G) = cod(G)$$

which says that "the codomain of the composition F.G of an arrow F and an arrow G (as far as they are composable) is the same codomain of G."

At a little bit right of the lower center of the figure is another sheet of thought with small proofs. This part proves that "from assumptions F=G and G=GG, the formula F=GG can be deduced"(i.e. the transitivity of = relation) by using the primitive inference rule "=repl-formula". Obviously this will be used as a derived rule.

In the lower window of Figure 7.15 is a proof of the theorem:

$$F : obj => F : iso$$

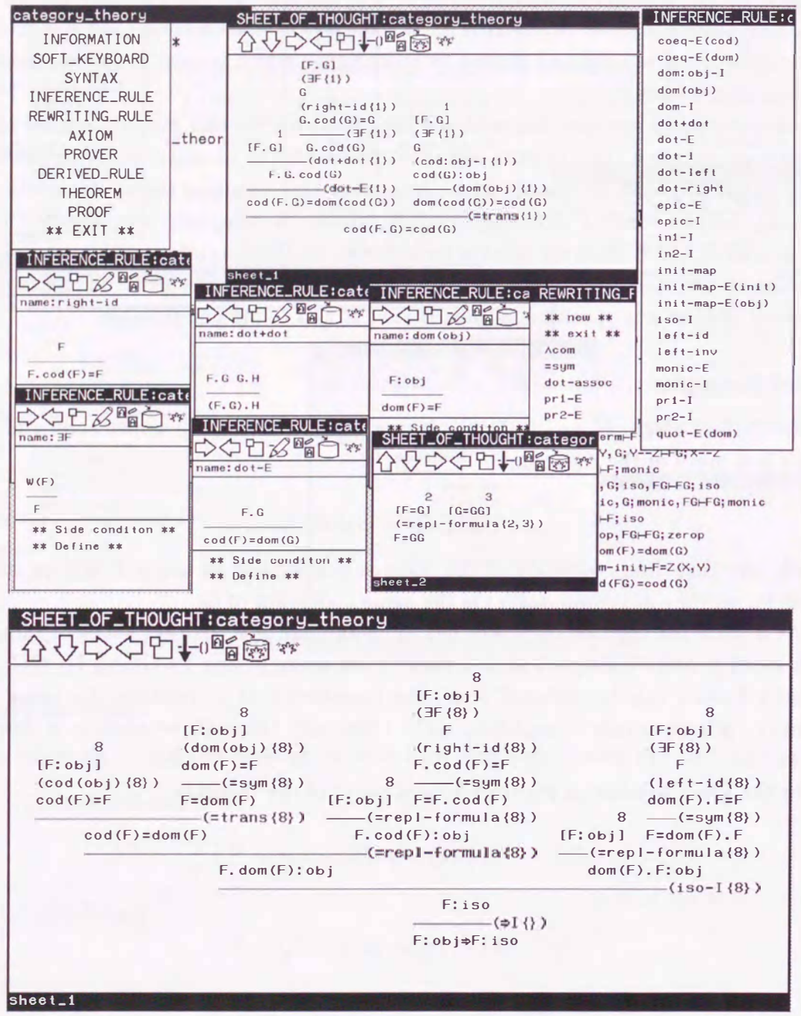


Figure 7.15: A Formulation and Proofs of Category Theory

Chapter 8

Application to Knowledge Acquisition Support Systems

The aim of this chapter is to present an application of EUODHILOS architecture to a new field which is considered to become very important in the future and indicate that it has wider application field other than pure logic-based reasoning. The application field dealt with in this chapter is knowledge management(KM) for a number of people such as in a research group, a department, a corporation, and so on, where each person is involved in formal reasoning.

We develop a framework of a knowledge acquisition support systems(KASSs) as an application of the reasoning assistant system framework of EUODHILOS towards the field of knowledge management.

We classify the knowledge data into two types: domain-knowledge and meta-knowledge[72]. The domain-knowledge is the knowledge that describes the target domain. In our framework, the domain-knowledge is the data that specifies a logical system, which consists of language system and derivation system descriptions. By meta-knowledge we means the data that are obtained in the reasoning and problem solving. In our framework, the meta-knowledge consists of theorems, derived rules, and tactics.

The most important difference of these two types of knowledge is that the former is given by the users whereas the latter are generated, in a sense, mechanically in reasoning steps. The validity of domain-data depends on the validity of the formal model which the user specifies. The meta-data, on the other hand, are only a consequence of formal systems and reasoning. Their evaluation are not by validation but by their usefulness.

We provide a model of creating and sharing domain-knowledge and meta-knowledge among a group of people, where these two kinds of knowledge are represented in a logical framework. The problem and its domain are described in a formal or logical system. The users solve problems by using an interactive general-purpose reasoning assistant system like EUODHILOS which provides assisting facilities based on the incubator model of reasoning. The meta-knowledge (i.e. theorems, derived-rules and tactics) is acquired in such reasoning. Each meta-knowledge data is shared among a group of users in assistance with helper agents that form an agent network, who locate in the background of the reasoning assistant system. The agent network shares not only the meta-knowledge but also their reputation or evaluation information. It enables the system to perform "social selection[69]" of knowledge which helps the users with selecting appropriate ones. This model gives a new perspective to semi- or fully automated knowledge acquisition, group reasoning assistant system, and knowledge management.

8.1 Background

It is going to be widely recognized these days that creating and having valuable and useful knowldge is crucially important for companies being creative[76]. From this recognition, growing number of companies are paying more attention to knowledge management[45] technologies and systems. However the bottleneck of knowledge acquisition[47; 74] still remains a big problem. How can we acquire useful knowledge efficiently? Without solving this problem, knowledge management systems will have only limited success in creating and having a good use of knowledge in a company and contribute to keeping the company being competitive and also in other kinds of groups of people for being active and creative. In this chapter, we investigate a model for creating and sharing logic-based domain- and meta-knowledge among a group of people based on the model of EUODHILOS. Following to this model it is easier to acquire and share knowledge than dealing only with the knowledge extracted from human brains. Thus it will be a good choice for the companies to start with constructing knowledge management systems that deal with logical knowledge.

In a logical framework domain-knowledge is represented as a logic or a theory that consists of a language and a derivation system, whereas meta-knowledge is generated in the proving process and acquired as theorems, derived rules, and tactics. The language system gives basic vocabulary for describing the problem domain. By the derivation system, we specify the ontological[75] structures and their relations. Meta-

knowledge is created in sheets of thought based on the incubator model of reasoning. The model for sheets of thought in this chapter is an extension to the one in other chapters in such a way that some number of different helper agents may be attached to the sheets and work together with the human user and help the user with incubation of ideas in reasoning. Users will participate in this process by giving goals or conjectures to be proved, creating new results by applying tactics, and so on. The meta-knowledge data obtained in such a process will be circulated and evaluated with a network of communication agents. They are socially selected according to their reputation in usefulness.

The rest of this chapter is organized as follows. In Section 8.2, we give an acquisition model for logic-based domain-knowledge and a model for creating and acquiring meta-knowledge from reasoning on the target domain. In Section 8.3, we describe the logical framework of the knowledge acquisition system, which is based on that of the general-purpose reasoning assistant system EUODHILOS. Then we present the organization of the knowledge acquisition support system. The major idea is also borrowed from the reasoning assistant system EUODHILOS. The sheets of thought play the central role in meta-knowledge creation. The model for this environment is similar to "chemical reactor" or incubator of ideas in reasoning. Agents will monitor how new meta-knowledge data are created and help the user with reason effectively. In Section 8.4, we present a mechanism where agents work cooperatively with other agents, circulate and share useful meta-knowledge data and evaluate them so that useful data are to be "socially selected." In Section 8.5, we conclude the discussion and suggest some of the important problems to be solved in the future research in this direction.

8.2 Models of Logically-Represented Knowledge Acquisition

In this section we present two models for knowledge acquisition processes. The first one is the model for domain-knowledge acquisition, where the knowledge is extracted from human brain, whereas the other is the model for meta-knowledge acquisition, where the knowledge is generated in the reasoning process. Note that the difference in this respect induces a great difference in usefulness of these two kinds of knowledge.

(1) Model of Domain-Knowledge Acquisition

We borrow the model of acquisition process of domain-knowledge extracted from human brain from the reasoning process model taken in EUODHILOS(Figure 3.1). This is the one normally considered as knowledge acquisition process, which is also the process of scientific discovery.

The process consists of four basic steps. We will explain what they are like according to the figure clockwise from the top part:

(1) **Observation:**
First of all the image comes into human mind through his or her observation and experience.

(2) **Formalization:**
This is the step of analyzing our mental image and trying to extract knowledge and represent it formally. In our logic-based approach we have to find out the fundamental vocabulary for representing the target domain, and the structural rules of the target domain. In other word, this is the step of constructing a logical model of the domain.

(3) **Deduction:**
Superficially the previous steps are sufficient for constructing logical models. However, we recognize, from our experiences, that it is far from finishing the process of model making because it is quite difficult to construct a sufficient model in the first attempt. So it will be necessary to check if the model is good enough. We make conjectures that should be satisfied in the model and do some deductions on the model to see if the conjectures are really provable. This is the step for it.

(4) **Verification:**
As has been pointed out, most of the cases of constructing a logical model, it is not sufficient in the first trial. What should we do if we find it is not sufficient in the test of the previous step? Together with the third step, we compare the behaviour of the model and that of the nature or the real domain. Then we move back to the first step of observing and think what is/are wrong with the current model. Then we repeat the whole processes again and again until we have a satisfactory model.

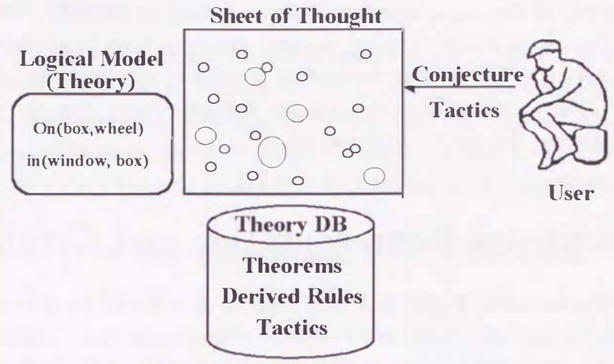


Figure 8.1: Meta-Knowledge Acquisition Model

(2) Model of Meta-Knowledge Acquisition

When the logical model is constructed, we will move to the problem solving phase, where we describe the problem in one or more logical formulas and solve it via proving the goal formulas. Meta-knowledge, consisting of theorems, derived-rules, and tactics, is produced in this reasoning process.

The model of acquiring meta-knowledge is illustrated in Figure 8.1. In this phase the proof-supporting facilities play the central role, which we call the sheet of thought. The user interacts with the sheet of thought by, for example, giving conjectures or goal formulas, tactics to apply to the proof fragments. On the sheet various proof fragments or partially constructed proofs are placed and waiting for being combined to grow up to be a complete proof. In the back of the sheet is the logical model that describes the target domain. The useful results of reasoning are stored in the theory database, where the created meta-knowledge data are saved, supposing to be used in the later reasoning processes.

Comparing to domain-knowledge, meta-knowledge has a great advantage in terms of acquisition:

(i) The meta-knowledge data are saved because the human user recognizes their usefulness and value; theorems are representing valuable results, and other two, derived rules and tactics, are considered to be useful in later reasoning. Therefore they are expected to be more useful than those chosen automatically by some criteria.

(ii) The validity of the meta-knowledge is guaranteed objectively. Theorems and derived rules have proofs of them. A tactic can be judged by applying the tactic to the proving situation; if it succeeds it is valid in the situation, if it fails it is invalid. Thus we can somehow automate the acquisition procedure for tactics. We will discuss this aspect in detail later.

8.3 Knowledge Representation and Creation

As has been illustrated in Figure 3.2, the data of EUODHILOS-II consists of two parts: the logic specification part and the proof construction part, which correspond to the two types of knowledge mentioned in Section 8.2. In Section 8.3, we deal with the data for the logic specification that are for representing the domain-knowledge. The data used in the proof construction part that relates to meta-knowledge will be dealt with in Section 8.3.

The logical framework of EUODHILOS is, unlike other well-known generic reasoning assistant systems[27; 29; 85], based on the natural deduction[89] style, thus easier to use for human reasoners, because:

(i) it is relation-based so that it is applicable to any style of reasoning, such as forward, backward, and mixed derivations, and

(ii) it allows assumptions for reasoning, so that it meets to the natural human reasoning style. We put special effort on usability[73] in designing EUODHILOS systems. The "usability" here does not mean just user-interface issues. We think the issues such as the system sufficiently meets the users' way of thinking or it is flexible enough to a wide variety of needs, are crucially important in usability issues.

(1) Domain-Knowledge Representation

In order to represent a domain-knowledge that corresponds to the logic specification part of Figure 3.2, we specify the language system first, then the domain with the derivation system that consists of axioms and one or more rules. The name of the logic corresponds to its target domain.

(i) Language System:

As was described in Chapter 4, a language system specifies the fundamental vocabulary for expressing the target domain. In our case it specifies the logical expressions such as formulas, terms, expressions, etc.. We may call this an ontology[75] or a knowledge representation. In our framework, users are able to specify what expressions to use according to their preferences by defining a language system.

(ii) Derivation System for Representing Domain-Knowledge:

As was described in Chapter 5, the derivation system consists of axioms, inference rules, and rewriting rules. In our current situation, it is desirable to have richer descriptions structures. We will leave this issue for the future. These rules are represented in a natural deduction[89] formalism, where users can perform assumption based reasoning. The domain structures represented in derivation systems are (i) logical structure, and (ii) domain-specific structure. There are a couple of choices depending on the skill level of the user. For a novice user, the system will provide a library of various logics and theories and the user just chooses what he or she wants. A medium-level user will borrow the fundamental logical structure from the library, then specify the domain-dependent knowledge by either modifying a theory in the library or setting all the additional structure description. An expert user who knows much about the description framework will specify all the structures by himself or herself, including logical structure.

By borrowing the framework of EUODHILOS systems, as was described in Chapter 5, our derivation system consists of axioms, inference rules and rewriting rules. An inference rule consists of a name, conclusion, one or more premises, and optional side conditions. Each premise may have one or more assumptions. An axiom is an inference rule that has no premises. A rewriting rule also resembles to an inference rule. However it is different in two aspects:

(i) it has only one upper formula that has no assumptions, and

(ii) it is applicable for rewriting any matching sub-expressions of a logical expression.

The framework allows users to attach one or more side conditions to the rules. It is sufficient to describe ordinary logical rules by combining the side conditions: `FREE-FOR`, `NOT-FREE`, `NOT-FREE-IN-ASSM`, `FULL-SUBST`, and `SYNTAX-CAT`.

The framework of the derivation system is, as has been pointed out in Chapter 4, strong enough for typical styles of formulation of logical structures. In the Hilbert-style formulation, the structure of the domain at hand is described by relatively a large number of axioms together with a small number of inference rules like modus ponens. In the sequent calculus formulation, most of the structural data are given as inference rules in which formulas are represented as sequents. Only the starting formula, for example, "$A \vdash A$", is given as an axiom. A term rewriting system has only rewriting rules. It is also possible to combine these three kinds of data. For example we can define a derivation system based on the sequent calculus formulation mixed with several rewriting rules. This framework is characteristic in its giving as much freedom for the formulation style to its users. Thus the users are free to choose the most suitable style according to their needs and preferences.

However it is still not good enough from the user interface point of view. There are many users who have little knowledge on logics and logical framework for representing target domains. We would need some kind of user interfacing facilities together with the primitive user interface described in this chapter.

Example: In this example we briefly illustrate how knowledge is represented in the logical framework. Suppose we have a job J which consists of tasks $t_1, t_2, \ldots, t_n$. One possible way of expressing this is as follows(or-parallel representation):

$$\frac{t_1 \quad t_2 \quad \cdots \quad t_n}{J}$$

This rule says that if we have done the tasks $t_1, t_2, \ldots, t_n$, we can say we have done the job J. In this formulation the tasks can be performed in any order.

If we want to express ordering relations among tasks, say these tasks $t_1, t_2, \ldots, t_n$ should be done in this order, we would write(and-parallel representation):
Axiom: t_1, and
Inference rules:

$$\frac{t_1}{t_2}, \ \frac{t_2}{t_3}, \ \ldots, \ \frac{t_n}{J}$$

Then we start with t_1 and do $t_2, \ldots, t_n$ following to this order until we have done J.

By combining these two types of representations we can specify both and-parallel, or-parallel and mixed ones of both styles of constraints.

(2) Meta-Knowledge Representation

Meta-knowledge is created in the reasoning process and selected by users. Theorems and derived-rules are the proved formula and reusable partial proofs, respectively. Tactics and tacticals provide a means to automate deductions. A tactic specifies the outline of a proof. A tactic takes a list of proofs and generates the list of proofs obtained by applying the proof procedure described in the tactic. A tactical is a metafunction over tactics; it combines one or more tactics and creates a new one. Using tacticals we can create useful tactics that are complicated enough to express our intended proof procedures. We have described for details in Chapter 6.

(3) Knowledge Creation

In this section we deal with single-user case of reasoning. EUODHILOS is the basic model of the front-end of the knowledge creation system. From this model, we have two phases of reasoning:

(1) describing the problem domain under consideration and

(2) the reasoning, or proving. In this section we deal with the user-interfacing and usability issues of such a reasoning assistant system.

(i) Setting the Underlying Logic

In this chapter we suppose the users themselves give definitions of logical systems. However in the actual application to each domain the style can be different according to the management policy. For example, it is possible to take a style where the project manager gives the definition of the target logic and other project members share this definition. Note that each member can add some extra definitions to the original logic and define his or her own theory. The theory created in this way can be used to be inherited for defining other theories as well.

Through the syntax definition window, the users can input their definition of logics to the system. Metavariables are used as place holders of various expressions. The representation style of logical structure of EUODHILOS is intuitive and easy-to-understand, because the rule is displayed in tree-form as are used in the standard textbooks of logics so that the users would be able recognize the relationship of the components at the first glance. The side conditions are given as a set of elementary

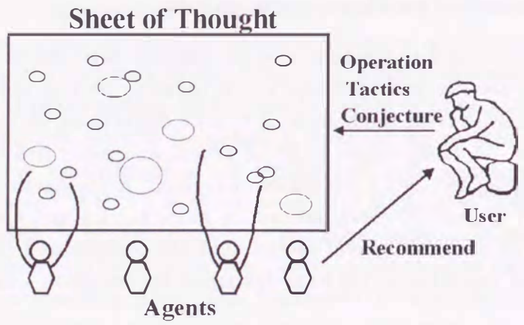


Figure 8.2: Incubator Model for Sheet of Thought

side conditions so that it enables the users define the rules in detail unlike other logical frameworks.

(ii) Sheet of Thought

A sheet of thought is a field where users create ideas for reasoning, where a number of elementary bits of information for reasoning, i.e. conjectures, axioms, derivation rules, and partial proofs, are located and wait for growing by being combined to be big blocks of proofs, and eventually to be whole proofs of theorems. We call this style of reasoning the incubator model, which is illustrated in Figure 8.2.

Unlike the model of sheet of thought in other chapters, in the model of this chapter, not only the user in the right side of the Figure 8.2 but also the helper agents that appear in the bottom area are monitoring how the components in the field are changing; we may call this a blackboard[22] model in this respect. Each agent has its own mission and monitors the field according to it. When it finds a pattern in its responsibility, it would take the components and create one or more new components by combining or dividing the components it has captured. The user controls the sheet of thought not only directly by putting conjectures, applying some tactics, and so on, but also indirectly by giving commands to one or more agents, assigning a new agent for additional facilities.

The commands given to an agent from the user include:

(i) confidence value; if an agent is highly confident or it has previous permission it would act automatically without asking to the user,

(2) adding or deleting action list of the agent; the action list specifies what sort of actions the agent can take, and others.

Note that the agents are not allowed to always act fully autonomously. Like Maxims[53] agents have three states for each action item; being confident and do the action automatically, having moderate confidence and give suggestions to the user, and lacking confidence and just observing.

The basic operations for proving performed on a sheet of thought are basically the same as those of the reasoning assistant system. We have described this issue in Chapter 6.

8.4 Sharing Knowledge with Agent Network

In the previous section we deal with the single-user case. In this section we consider the multi-user case; we propose a model of circulating and using the meta-knowledge data created in the reasoning processes. In this model, useful meta-knowledge data are distributively and selectively shared among the users. "Selective" here means that only the data being estimated as useful for a user are delivered to the user. So the data are not shared automatically among all the users. We will propose and illustrate overall structures of this mechanism.

8.4.1 Circulating and Sharing of Knowledge

In Section 8.3 the knowledge acquisition system for single-user case was illustrated, where the sheet of thought plays an important role. Various kinds of agents may be attached on it. They will watch the sheet and capture the specific proof patterns they are supposed to deal with. According to the given mission each agent will modify one or more proof fragments with, for example, combining these proof fragments, dividing proof fragments, giving suggestions or recommendations to the user, and so on. The resulting theorems, derived rules, and tactics are stored to the theory database, which are chosen by the user.

In this section we propose a model, which is illustrated in Figure 8.3 in which every knowledge acquisition system has a special helper agent called the communication agent, who takes communications with other communication agents. With the communications the useful meta-knowledge data and their evaluation data are circulated and shared among the network agents selectively. By utilizing this network, the users are able to use useful meta-knowledge provided from others. In this way they

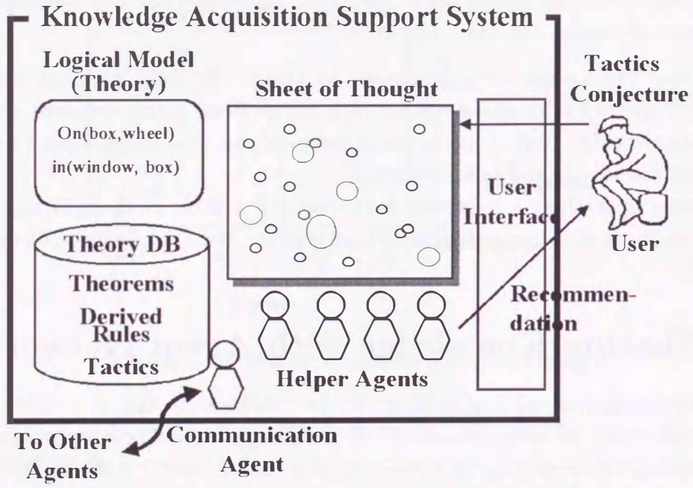


Figure 8.3: Knowledge Acquisition Support Sytem

form a good cooperative relationship and each participant enjoys the benefit of this cooperation.

Each user has a reasoning assistant system which we have explained in Section 8.3. The system communicates with the user. In the back of the systems are agents who are connected each other and form a network. Each agent monitors on the sheet of thought and waits for the new meta-knowledge data being created. When it finds a new knowledge data, firstly it investigates the new data; the data type, the background theory, what logical rules it includes, and others. Then it will send the data to its appropriate neighbouring agents.

For the networking mechanism suitable to such a purpose, we have proposed a mechanism called "word-of-mouth agent system"[82], in which we will have the benefits:

(i) The information data exist and are managed distributively so that the access traffic will be less concentrated than managing them in a central server. The system will also be more robust.

(ii) Circulated information is located in a limited area at first. Then the area expands adaptively according to the needs for the information.

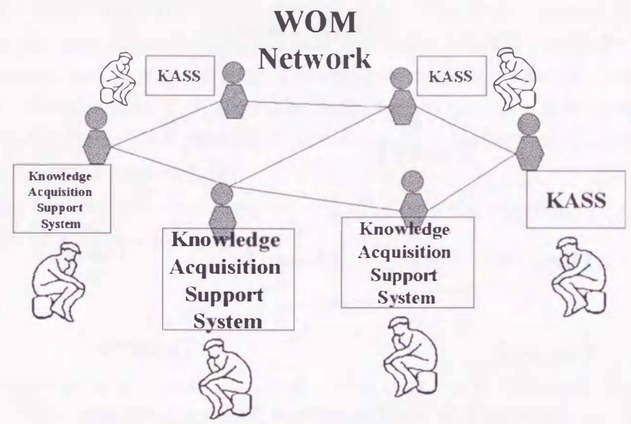


Figure 8.4: Word-of-Mouth Agent Network

(iii) The evaluation data are also circulated together with the information resources so that it is easier to choose appropriate ones from among a lot of candidates.

Figure 8.4 illustrates that the communication agents are connected each other in a relatively small number of connections and form a WOM network. The meta-knowldge data obtained in a knowledge acauisition support system(KASS) will be transmitted to its neighbours. The area where the data are transmitted is relatively small at first, then it expands its area according to the needs and reputation to it. If the agent gets this data and finds it useful by using it, the agent will circulate the data further. Such a mechanism will make the meta-knowledge circulate in a wide area if its reputation is high. We will call such phenomenon "Social Selection." Such a mechanism does not appear in other systems where they use the evaluation data given from the users[21].

In order to cope with sharing knowledge among agents, it is necessary to translate the language system(ontology translation). It is relatively easy for logical expressions, because they have similar structures comparing to the domain-dependent knowledge, with which this translation problem is as very hard as that of the other types of knowledge representation methods and their ontology[75] translations. We will not discuss this type of translation in this thesis. Instead, we would put more focus on translation of derivation systems. The derivation systems are more or less similar to

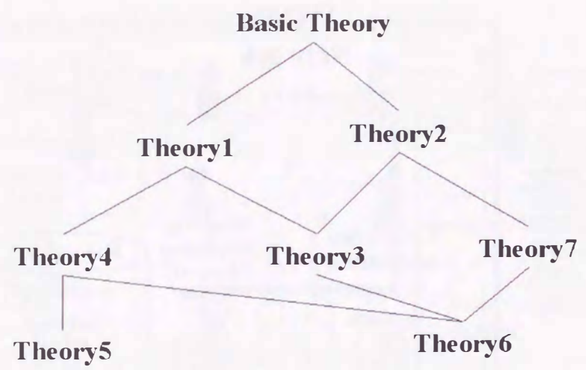


Figure 8.5: An Example of Theory Hierarchy

each other comparing to other types of logical specifications. However it also has a translation problem between different style of formalisms. Suppose we take different types of formalisms of logics. There are three typical types of formulations; Hilbert style, natural deduction, and sequent calculus. Generally speaking it is impossible to convert between these types of formulation. In order to cope with this problem, we will prepare a library which contains typical types of conversion procedures. In the future systems, it will become practical to have a theory for translating between different formalisms. Suppose some user develops a theory and proves a theorem that justifies the validity of a translation algorithm, then the system allows the users to use this algorithm for translating data from one theory to another.

One of the biggest advantages of this mechanism is that as the user works hard and creates a lot of reasoning results he or she would appreciate the useful meta-knowledge data. Also the created results will benefit other users. The amount of results created in reasoning and that of benefits for reasoning obtained from others are strongly correlating each other. This indicates that in our mechanism we will have little or no "free-riding problems", which is a big problem for many recommender/collaborative-filtering systems[92].

8.4.2 Generalization of Meta-Knowledge

Theories may be created by inheriting one or more old theories or logics, thus we have an ordering structure of theories we are dealing with in a group of users.

Figure 8.5 is an example of a theory hierarchy. The Basic Theory is the original one of all the theories. Theory1 and 2 inherit the Basic Theory. Similarly Theory4 and 7 inherit Theory1 and 2, respectively. Theory3 inherits both Theory1 and 2. The rest theories are created similarly. It would be more understandable if we take elementary set theory for the Basic Theory, algebra for Theory1, topology for Theory2, algebraic topology for Theory3, and so on.

Using this hierarchical structure, we will illustrate the algorithm for generalizing the used area of a meta-knowledge data.

Step 1:

Suppose we have a tactic created by a user who works with Theory6. It is applicable to Theory6. The system would think the tactic may be applicable to other theories besides Theory6. Firstly it will check out all the axioms and rules that are involved in the tactic. If all the components included in the tactic are of Theory4, it means that it is well applicable to Theory4.

The algorithm goes up in the hierarchy until it finds all the maximal theories that the tactic is applicable. Suppose Theory1 is the only maximal theory.

Step 2:

In the next step, the algorithm goes down from the maximal theories and collects all the theory names. These are all the theories that the tactic is applicable. As we are supposing Theory1 is the only maximal theory, the theories the tactic is applicable are Theory1, 3, 4, 5, and 6.

Step 3:

Finally, the algorithm finds out which user uses at least one of the theories in the list. Then the network agent of the theory sends the tactic to the network agents of the specified users.

This mechanism is expected to create better tactics than just creating them randomly, because only those tactics that have good reputation in at least one theory are recommended to other theories.

8.4.3 Social Evaluation and Selection of Meta-Knowledge

As was briefly mentioned in the previous section, dealing with evaluation or reputation data would be one of the key features for having a good use of knowledge and meta-knowledge in a group of people or agents. The meta-knowledge data such as tactics, theorems, and derived rules that are applicable to the sheet of thought, are gathered to its communication agent. They are applied to the sheet of thought. If they succeed they will get high evaluation values(i.e. good reputation), whereas if they fail they will be evaluated low. The meta-knowledge data which is of high reputation would be circulated among a wide range of agents and thus will survive in the network. On the other hand, the meta-knowledge having poor reputation will not be used by any agent and will be disappeared in the long run. We call this mechanism "social selection[69]."

This mechanism can be effectively applied to finding useful meta-knowledge data because meta-knowledge are "testable" and thus safe to be applied. We cannot apply it to domain-knowledge. Suppose we have a domain-knowledge "$bird(x) \supset fly(x)$" in Singapore. It is also strongly supported in Hong Kong, in Japan, in America, in Europe, etc. etc.. Eventhough we cannot apply this domain-knowledge to Antarctica, where it is not true and "$bird(x) \supset swim(x)$" may be the most important knowledge about birds there. This difference comes from that domain-knowledge gives starting knowledge about the domain, thus we have to add new knowledge or delete old ones with much care. It directly affects to the validity of the domain model. On the other hand, using meta-knowledge does not affect to the validity of the model. A non-theorem cannot be a theorem even if we want to do. The usefulness of a tactic can be verified only after we apply it to the actual proofs.

An agent gets a meta-knowledge data with information such as which agent evalutes it with what value. The agent then estimates its usefulness in the domain he is dealing with. He may have a number of estimation functions. Considering the estimated value of these functions and the evaluation values from other agents, the agent calculates its initial evaluation value for the meta-knowledge. This value will be modified after the knowledge is applied to various proof constructions. The agent also estimates usefulness for his neighbouring agents. If he is confident on usefulness of a meta-knowledge for another agent, i.e. the estimated usefulness value is greater than a value determined previously, then the meta-knowledge will be introduced to the neighbouring agent. By repeating this process meta-knowledge data are selected socially; meta-knowledge with good reputation will be distributed widely and be used

in many times.

8.4.4 Modification and Composition of Tactics

In order to produce a variety of useful meta-knowledge, we provide a means to create tactics by modifying and combining those tactics we already have. The generated tactics are evaluated and useful tactics are selected in the social selection mechanism described in the previous section. We will show some example methods of creating new tactics. Firstly, we present the definition of tactics.

(a) Definition:

A tactic is constructed in either one of the (i) primitive tactics: axioms and rules, or (ii) combined tactics: the tactics of the form $T(t_1, t_2, \ldots, t_n)$, where $t_1, t_2, \ldots, t_n$ are tactics and T is a tactical with arity n.

Tacticals are the primitive functions that generate tactics by combining one or more tactics. Some of the most popular tacticals are `REPEAT`, `THEN`, `ORELSE`, and so on.

(b) Modification:

Here are two possible modifications:

(i) $t \Longrightarrow t' =$ `REPEAT` t, and

(ii) $t = t_1$ `THEN` $t_2 \Longrightarrow t' = t_1$ `THEN` t_2 `ORELSE` $(t_2$ `THEN` $t_1)$

(c) Composition:

$t_1, t_2 \Longrightarrow t_1$ `THEN` t_2

These algorithms create tactics from the tactics that are considered useful (in a theory or more of them). It would be reasonable to assume those tactics generated in these algorithms are expected to be more useful than those generated from arbitrary tactics.

The new tactics will be used and be selected in the social selection mechanism that we have illustrated in the previous section. In these processes tactics will be evolved in the agent network. This is important because it gives a mechanism for automatically acquiring useful knowledge.

8.5 Chapter Summary

We have proposed a model of creating and acquiring logic-based domain-knowledge and meta-knowledge. Domain-knowledge is represented in a logical framework by borrowing the EUODHILOS architecture, in which rules are represented in a tree-from based on the natural deduction style formalism. The model of domain-knowledge acquisition is the one for creating a logical model of the target-domain, which is, in nature, through repeating trial and error processes. Meta-knowledge is originally created in reasoning and chosen as such by the users. The reasoning process, which we call the incubator model of reasoning, is performed in the sheet of thought, on which small proof fragments grow to be complete proofs as the reasoning advances. Users participate this process by setting goals, manipulating proof fragments, applying tactics, and so on. Some helper agents also work on the sheet. They monitor the situation and help whenever they find their helping patterns. The resulting meta-knowledge is circulated among the communication agents. The circulating meta-knowledge data are applied to the sheets of thought on which they are expected to be useful. The results of application is evaluated and such reputation information is also circulated and used in the recommendations to the users. This application and evaluation mechanism is called social selection.

In dealing with the logic-based knowledge we have advantages such as:

- Not only the ordinary manual knowledge acquisition, we are also able to create and use automatically-acquired knowledge,
- We can deal with both manual and automatic evaluation data so that we can combine them appropriately and realize the social selection mechanism, and
- Logic-based knowledge can be easily used combined with the systems and algorithms developed in the research on logics including theorem proving, automated reasoning, and so on.

The characteristic features of the incubator model include:

- Agents automatically assist various patterns of reasoning. The user can easily customize the assisting feature by modifying the helper agents.
- In this model, the communication agents work cooperatively and circulate the useful meta-knowledge. It is a great advantage that as the user work hard on his or her reasoning, the user would have more knowledge data from the network,

thus it will benefit to his or her reasoning. At the same time, the more amount of knowledge will be created and thus contribute to the reusing of knowledge as well.

- Such a flexible model can be realized due to taking the relation-based flexible logical framework and its supporting environment by the sheets of thought.

With these characteristic features the models proposed in this chapter will give a new perspective to (semi-)automated knowledge acquisition, group reasoning assistant system, knowledge management and others.

Comparing with other reasoning systems[5; 9; 15; 19; 27; 29; 42; 85], we believe EUODHILOS is the most suitable one for reasoning in such a situation described in this chapter, because for this purpose the system should take a good balance as a logical system and an interactive user-oriented one; which we have been taking much effort in designing EUODHILOS systems.

Important future plans toward the application of EUODHILOS architecture to knowledge acquisition support systems include:

(i) Investigating corporate knowledge management systems, where the system deals not only logic-based knowledge but also other types of knowledge such as rules in a corporation, the data on human skills and human networks, and so on.

(ii) In this chapter we presented a simple example of sharing and reusing tactics. This mechanism needs to be developed further so that it deals with a lot wider matchmaking.

(iii) The interface to the users in this chapter is very simple. For more practical applications the reasoning environment should provide more easy-to-recognize view by using more sophisticated visualization algorithms.

(iv) Implementing and evaluating of automatic tactics improvement mechanism are another important topics to be pursued.

(v) So far, by using a general-purpose reasoning assistant system EUODHILOS, as was illustrated in Chapter 7, we have experimented with a variety of logics such as: NK, a constructive type theory, Hoare logic and Dynamic logic that are logics for programs, intensional logic, linear logic, genral logic, relevant logic, category theory, and others[100]. So the framework for defining logical structure is general enough for dealing with various logics. However in the

current interface, the users should define, read, and deal with the "raw logical representations." In order to be used by less logic-oriented users, we have to investigate more advanced system that does not look so much like a logical system.

Chapter 9

System Comparison

The aim of this chapter is to shed light on the characteristic features of EUODHILOS systems in comparison with related systems together with the comparison between themselves.

Much work has been devoted to building up the systems for checking and constructing formal proofs in various logical systems, e.g. see [3; 43; 111; 114] for proof checker, see [15; 27; 96] for proof editor/constructor, see [16; 30; 32; 64; 97] for general system of computer-aided reasoning. See Chapter 2 for their natures in terms of their assisting styles to reasoning. In this chapter we will rather confine ourselves to various approaches to the general system for computer-assisted reasoning to which much attention have been recently paid.

We start with classifying the styles of representation of logical systems and making clear what our approach is like in comparison with other approaches. We take account the reasoning systems that allow the user to somehow define the logical systems here and classify them into the following three categories according to their approaches:

(i) Implementing logics in functional or logical programming languages

The first approach is to implement a logical system by using a programming language. This is general in the sense that whatever systems can be implemented in such a generic programming language. However, by considering the implementor's burden for programming, it is reasonable to confine ourselves to the functional and logical languages which are supposed to be suitable in our situation. In [96], Prolog is employed as a logic description language as well as an implementation language of a proof constructor. The underlying programming language for specifying a variety of logical structures including higher-order logics, it is preferable that the implemen-

tation language itself is higher-order; otherwise it is quite difficult for the users to implement their intended systems. Thus HOL[29] has, as is easily guessed from the name, a higher-order logic as the underlying programming language. In [23] and [59], λ-Prolog, which is an extension to Prolog with the higher-order facilities based on λ-calculus and hence more expressive than Prolog, is proposed to specify theorem provers. In [90], 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 [28], the metalanguage for interactive proof in LCF[27], a polymorphically typed, functional programming language, is used to show how logical calculuses can be represented and manipulated within it.

(ii) Encoding logics into a formal system

The second approach is based on a metalogic and its manipulation functions. The object logics are represented and manipulated through the metalogic. Such systems have the advantage that the object logics have rigorous semantics based on those of the metalogics. Thus they are suitable for developing theories by extending the object logics. However, the users are required to have specific knowledge and skills to describe the logic properly, hence these systems are mainly intended for experts of logics, preferably, higher-order logics. The purpose of Nuprl is very similar to that of EUODHILOS. It aims at providing the proof construction environment for a variety of logics. But the approach to the realization of it is different to those of the reasoning assistant systems. Nuprl has a fixed underlying logic and other logics must be defined by using the terms of this logic. In the approach of reasoning assistant system, even the syntax of the logic is expressed by the user. It aims at the complete realization of logic-independent systems which can assist human reasoning in the various fields. Proof construction methodologies are also different. In Nuprl, proofs are constructed only by refinement, while in our system, proofs are constructed by three types of deductions; in forward, backward (same as refinement), and filling the gap in proofs. In [30] and [32], a typed λ-calculus with dependent types is used for building a logical framework 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 [84] and [85], a logic is encoded to a subset of a higher-order logic. What they are aiming principally at seems to be automatic checking of rule conditions basically in one way reasoning, with which we are confronted in applying

a rule. In [86], a logic is to be encoded to a subset of a higher-order logic.

(iii) Representing logics so as to directly reflect their proof theoretic nature

The third approach to a general reasoning system is quite different from other two. The systems in the second approach provide a higher-order logic as the metalogic and users can define the target logics in this logic. Therefore the expressive power is quite strong and it is fairly easy to prove the justification of the representation. However the description of logical systems are usually complex and also in many cases the proof supporting facilities are not strong enough. Contrarily, the systems in the third approach do not assume metalogics as their underlying logics. It allows users to specify logics in an easier and more direct way than others which require them to learn a programming language or metalogic for encoding a logic. Moreover, it provides reasoning facilities and a unique reasoning-oriented interface to make proof construction more flexible and easier. As a result, these systems provide novice users with extensive assistance. Besides EUODHILOS systems[67; 80], ωple[19], MURAL[42], PROOF DESIGNER[5] take this approach.

As is displayed in Table 9.1, we compare EUODHILOS-I, EUODHILOS-II, MURAL, PROOF DESIGNER, and ωple with the viewpoints of syntax definition framework, proof style, proof representation method, derivation systems, implementation language, and platform. From this table we read that EUODHILOS systems provide the facilities for dealing with a much variety of representation styles of proofs and a variety of reasoning assisting features. The new symbol definition and inputting facility and candidate selection features are characteristic to EUODHILOS systems.

The proof style of EUODHILOS and MURAL are based on natural deduction, ωple is based on the sequent calculus, and the proof style of PROOF DESIGNER is based on the Fitch style. Proofs are represented in tree-form on EUODHILOS and ωple. MURAL and PROOF DESIGNER use a line-based representation. Only EUODHILOS and MURAL allows the mixed reasoning style. ωple allows only backward reasoning, and PROOF DESIGNER allows only forward reasoning. These comparisons show that EUODHILOS systems provide more flexibility in proof representations and proof styles than the other systems.

To conclude, the approach of EUODHILOS to a general-purpose reasoning assistant system(G-RAS) differs from the other ones cited above in several respects.

- In EUODHILOS one can specify his or her own logic in a more direct and tractable way than others which require us to learn a formal system or metalogic

Table 9.1: Comparison of EUODHILOS Systems with Similar G-RAS Systems

System Name	EUODHILOS-I	EUODHILOS-II
Syntax Definition	DCGo	Context-Free Grammar
Proof Representation	Natural Deduction Style	Natural Deduction Style
Proof Display Method	Tree Structure	Abridged Form Line Form Tree Structure
Derivations	Axiom Inference Rule Rewriting Rule Theorem, Derived-Rule	Axiom Inference Rule Rewriting Rule Theorem, Derived-Rule
Proving Methods	Forward Derivation Backward Derivation Mixed Derivation Connection	Forward Derivation Backward Derivation Mixed Derivation Connection Proof Tactics
Implementation Language	ESP	Emacs Lisp
Platform	PSI/SIMPOS	GNU Emacs/X Window

System Name	MURAL	PROOF DESIGNER	ωple
Syntax Definition	Context-Sensitive Grammar	Context-Free Grammar	Context-Free Grammar
Proof Representation	Natural Deduction Style	Fitch Style	Sequent Style
Proof Display Method	Line Form	Line Form	Tree Structure
Derivations	Axiom Inference Rule Fold/Unfold	Inference Rule	Inference Rule Derived-Rule
Proving Methods	Forward Derivation Backward Derivation Mixed Derivation Proof Tactics	Forward Derivation	Backward Derivation Proof Tactics
Implementation Language	Smalltalk-80	Lightspeed Pascal	ω-Prolog
Platform	UNIX	Macintosh	UNIX/X Window

for encoding a logic.

- Much emphasis has been placed on reasoning facilities and proof methods which EUODHILOS should have in order to make proof construction more powerful and easier. In other works on interfaces for theorem provers(e.g. [6]), they put major emphasis on the general interface issues. The interface of EUODHILOS systems were designed by considering the characters of the platforms and the general policy of EUODHILOS architecture. We believe this methodology of system design contributes to the high usability of the system.

- The system architecture can be evaluated based on such a model-based methodology[73], where the system evaluation items were extracted according to the model of user's reasoning and conception process. This methodology will contribute to a more objective and comprehensive evaluation than evaluating only in an empirical method[39]. Therefore such an approach should get more attention for evaluating user interface in order to have a more comprehensive evaluation of reasoning assistant systems and other types of systems.

- EUODHILOS has a unique reasoning-oriented interface not only for raising user-friendliness but also helping us conceive ideas for constructing the proofs. Dawson's generic logic environment is very similar to our approach in many ways, but it only deals with logics in sequent presentations with all-introduction rules.

As has been described EUODHILOS-I and EUODHILOS-II systems share the common fundamental design model and policy of EUODHILOS architecture. For example, plain language specification framework, derivation system definition facilities based in the natural deduction style, flexible reasoning styles that meet the styles of human reasoning, reusing proof results with theorems and derived rules are their common features. Some aspects in the implemented systems are different caused from the differences of the implementation platforms and specific designing policies. They include the following issues:

- Syntax Specification Framework:
  In EUODHILOS-I, syntax of logical expressions are specified in the DCGo notation, whereas in EUODHILOS-II, they are specified in an BNF-based notation. In the former notation, some context sensitive syntactic constraints can be specified by using the arguments of nonterminals. In the latter notation, one may give specifications for binding variable and its scope in a flexible way.

- Validity Checking Facilities:
  EUODHILOS-II is equipped with richer facilities for checking for the matching of the derivation data and proof data to language definition.

- Proof Abridgment:
  In EUODHILOS-I, the use of lemmas and derived rules mainly contribute to building up smaller proof representations. It also support the multiple applications for rewriting rules. EUODHILOS-II extends this framework and provides the semi-automated proving facility with tactics/tacticals without providing detailed parameters such as assumptions, application place, and others.

- Proof Representation:
  In EUODHILOS-I, proof fragments are displayed in full-tree form, whereas in EUODHILOS-II, they are represented in two forms; full line-based tree representation, and that represented in LaTeX macros.

- Platform:
  The platforms are different. EUODHILOS is implemented on the PSI machine with SIMPOS operating systems. It runs on the SIMPOS window systems and was designed to have a good use of this environment. The platform of EUODHILOS-II, on the other hand, is Emacs that runs on UNIX machines and other popularly used machines, so that it has high portability.

Chapter 10

Conclusion

The aim of this chapter is to summarize what we have achieved in this thesis and to give possible future research directions that will become more important in the coming years.

(1) Achievements

In this thesis we have investigated a general-purpose reasoning assistant system architecture with which users are able to define their own formal systems in logic-based framework and reason on the defined formal system. In order to investigate such an architecture we took the model-based approach, where we firstly made the model for human reasoning process, then we established an abstract system architecture called EUODHILOS. We have developed two systems on different platforms by instantiating each components of the general architecture; EUODHILOS-I on PSI/SIMPOS and EUODHILOS-II on top of GNU Emacs/Epoch/Mule which runs on the UNIX machines. By taking such an approach we are able to clearly recognize the features that depend on the implementation platforms and those that come from the common architecture. For EUODHILOS-I, we chose the mouse-based menu-oriented design that fitted to the graphical window system of the SIMPOS operating system. For EUODHILOS-II, on the other hand, we chose the keyboard-oriented user-interface design where the underlying environment is the GNU Emacs, which is well known as a text editing environment.

Through various experiments of applications on a variety of logical systems and formulations, we have demonstrated the potential and usefulness of EUODHILOS systems. From these experiments we are convinced that the current EUODHILOS systems are fairly useful in two senses:

(i) As have been the target, EUODHILOS systems are suitable to experiment with a wide styles of logical formulations. They are especially suitable to learning-by-experimenting style of reasoning, thus they should be well applicable to computer-aided learning for logical systems.

(ii) Relating to the previous item, they are fairly suited as an assisting system for developing formal systems, especially in logic-based approaches. As the model of human reasoning process in Figure 3.1 suggests, the human user needs some amount of experiences before finishing the logical model construction. EUODHILOS system is one of the best choices for the user to learn his or her own logical model under construction.

We have also demonstrated how EUODHILOS systems can be applied to knowledge management. EUODHILOS systems together with agent technologies, the logic-based knowledge, especially the meta-data such as tactics and theorems fairly suit to be shared among a group of people. The evaluation issues on potential and usefulness of the system has also discussed in other papers[73; 100; 103; 104].

The issues what we put special emphasis on in this thesis are:

(a) Advantages of Generality:

The generality of EUODHILOS have been tested by using it to define various logics and to construct proofs expressed within them. All the logics with their proofs were created in several hours. If we had had to develop a reasoning system with the same functions as EUODHILOS for each logic from scratch, it would have taken much more 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 much wider fields of applications[81; 100].

(b) Flexible and Easy-to-Use Logical Framework:

The logical framework, which gives the means to the users of expressing what logical or formal system they want to deal with. In EUODHILOS-I, it is based on the DCGo notation, which is an extension to DCG(Definite Clause Grammar) syntax description framework, and natural deduction style derivation system. In EUODHILOS-II, it is based on the ordinary CFG(Context Free Grammar)-based production rules with BNF(Backus-Naur Form) notation, together with the natural deduction

style derivation system. We have demonstrated that by using such logical frameworks a variety of logical structures can be dealt with in natural and easy-to-use fashion. We have presented a variety of examples like intuitionistic logic, modal logic, Hoare logic, category theory, relevant logic, combinatory logic, and other different types of logics, and have demonstrated that the framework is really applicable to such a wide variety of formal systems.

(c) Proving Methodology based on Sheets of Thought:

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

It is worth noting that what we have achieved in this thesis are:

(i) to propose a new system concept and its architecture based on a model-based approach,

(ii) to demonstrate its feasibility by implementing systems on different platforms, and

(iii) to verify its generality and usability through experimenting with many applications of the systems to a wide variety of logical systems.

We would also like to note here that the source codes and documentations of EUODHILOS-I are accessible at the ICOT Free Software(IFS) site in the URL:

`http://www.icot.or.jp/AITEC/IFS/IFS-abst/028.html`

and the latest version of EUODHILOS-II and its sample logics together with manuals, can be downloaded from the following URL:

`ftp://ftp.fujitsu.co.jp/pub/isis/euodhilos2`

(2) Future Directions

Considering the development of information technology, especially the networking environment these days, the most important topics to be pursued for the future directions of EUODHILOS systems are:

(i) Extending the Application Fields:
Knowledge management is one of the promising fields for applying G-RAS systems like EUODHILOS. In this field, most users will use such a system not for dealing with logics or logical structures but for problem solving and decision making by collecting a lot of knowledge and data from other systems in the network. In order to adapt to such an environment, EUODHILOS systems need to extend themselves in the method of dealing with and of representing the knowledge. We have to extend the current system model which is designed as a pure G-RAS system so that it is able to deal with the formulas and proofs with less-logical representations and less-logical styles of manipulations. By taking the user interface in such style, even the users who have little knowledge about logics will be able to use the system.

It is also possible to develop a new application field by extending the data from the formalized logical one to less-logical type of data, the G-RAS system would be used in wider area of reasoning, idea creation, computer supported cooperative work(CSCW), and others. We have been working on a system called ZK(Zeichen blocK)[70], which deals with not only the text(verbal) data but also other types of data such as the arrangement data and the picture data (non-verbal or analogue data). Such a system would bridge the gaps between the formalized reasoning in logical framework and the intuitive reasoning that deal with the pre-formalized or image-based information. The future system in this direction will give an environment to the people as a sharing tool for such hybrid information that consists of verbal and non-verbal data.

(ii) Investigating more Automated Reasoning:
A wide variety of reasoning styles should be supported by G-RAS systems. We have described in this thesis that the reasoning assisting facilities of EUODHILOS-I is implemented with putting emphasis on supporting the step-by-step, or manual, reasoning. The built-in tactics and tacticals are introduced in EUODHILOS-II so that the system supports from step-by-step reasoning to semi-automated reasoning. By using them the medium-sized straightforward reasoning can be performed quite easily. As has Kowalski pointed out[46], a program can be seen as "Logic+Control". From the view point of G-RAS, this statement can be interpreted as "Problem Solving=Model + Tactics", where the purpose of an algorithm is to solve one or a set of problems, model is the logical model, and tactics describes the control part of problem solving. By considering this observation and the previous discussions, the development of useful tactics and sharing in a network would open up a new programming field, which we call the "reasoning programming"[71], where the users develop descriptions or instructions that tell how to develop theorems or find solutions of the given problems. The descriptions can be interpreted as programs in the sense that they can be, basically, executed automatically. A reasoning program consists of a formal model and its instruction, or a tactic of it. These different information works together as a description of procedures. This style resembles to the programming environment of applets running on browsers. The whole browser window consists of HTML description and the ordinary programs, which may further consist of Java script and Java applet working together. Considering the complexity of environment on which the programs are executed, only one programming language would not be sufficient. So the programming style seems to be going to the new style, where, like the browser programming and reasoning programming, a couple of programs written in different programming languages work cooperatively as an integrated single program. This programming model is also similar to that of agent programming, where different agents work together and form a system. Investigating the automated tactics programming framework will contribute to such a new paradigm of programming.

(iii) Developing Multi-Theory Environment:
In Chapter 8, we mentioned "ontology transformation". The knowledge and data we are using becomes larger and larger these days. The technology for integrating knowledge and data is also becoming more and more important. The investigation on theory revision and theory inheritance or transformation relate to the investigation on ontologies in its essential part. This is another important direction of research that relates to EUODHILOS, or G-RAS systems in general.

Bibliography

[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, Do-it-Yourself Type Theory(Part 2), *ibid.*, No.35, pp.205-245, 1988.

[3] Barwise, J. and Etchemendy, J.: A Situation-Theoretic Account of Reasoning with Hyper-Proof (Extended Abstract), *STASS Meeting*, 1988.

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

[5] Bedau, M. and Moor J.: PROOF DESIGNER: A Programmable Prover's Workbench, *Philosophy and the Computer*, Westview Press, pp.218-228, 1992.

[6] Bertot, Y., Kahn, G. and Théry, L.: Proof by Pointing, *Theoretical Aspects of Computer Software*, *LNCS*, Vol.789, Springer-Verlag, pp.141-160, 1994.

[7] Boyer, R. B. and Moore, J. S.: *A Computational Logic Handbook*, Academic Press, 1988.

[8] de Bruijn, N. G.: The Mathematical Language AUTOMATH, its Usage and Some of its Extensions, Laudet, M., Lacombe, D., Nolin, L. and Schutzenberger, M. (eds.), *Symposium on Automated Demonstration*, Springer-Verlag, 1970.

[9] de Bruijn, N. G.: A Survey of the Project Automath, Seldin and Hindley (eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda calculus and Formalism*, Academic Press, pp.579-606, 1980.

[10] Burkholder, L.: The Halting Problem, *SIGACT NEWS*, Vol.18, No.3, pp.48-60, 1987.

[11] Chang, C.-L. and Lee, R. C.-T.: *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, 1973.

[12] Chikayama, T.: EPS Reference Manual, ICOT Technical Report, TR-044, ICOT, 1984.

[13] Clocksin, C. F. and Merish, C. S.: *Programming in Prolog*, Springer-Verlag, 1981.

[14] Constable, R. L., Johnson, S. D. and Eichenlaub, C. D.: An Introduction to the PL/CV2 Programming Logics, *LNCS*, Vol.135, Springer-Verlag, 1982.

[15] Constable, R. L., et al.: *Implementing Mathematics with the Nuprl Proof Development System*, Prentice-Hall, 1986.

[16] Coquand, T. and Huet, G.: Constructions: A Higher Order Proof System for Mechanizing Mathematics, *LNCS*, Vol.203, Springer-Verlag, pp.151-184, 1985.

[17] Dawson, M., Sadler, M. and Mainbaum, T.: Generic Logic Environment, *Proc. of CASE'88*, pp.215-218, 1988.

[18] Dawson, M.: Using the ωp Generic Logic Environment, Technical Report, Dept. of Computing, Imperial College, 1989.

[19] Dawson, M.: *A General Logic Environment*, PhD thesis, Dept. of Computing, Imperial College, 1991.

[20] Earley, J.: An Efficient Context-Free Parsing Algorithm, *Communications of ACM*, 3(2), pp.94-102, 1970.

[21] Edwards, G., Kang, B.H., Preston, P. and Compton, P.: Prudent Expert Systems with Credentials: Managing the Expertise of Decision Support Systems, *International Journal of Bio-Medical Computing*, Vol.40, pp.125-132, 1995.

[22] Erman, L.D., Hayes-Roth, F., Lesser, V.R. and Raj Reddy, D.: The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty, *Computing Survey*, Vol.12, pp.213-253, 1980.

[23] Felty, A. and Miller, D.: Specifying Theorem Provers in a Higher-Order Logic Programming Language, *LNCS*, Vol.310, Springer-Verlag, pp.61-80, 1988.

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

[25] Gallin, D.: *Intensional and Higher-Order Modal Logic, with Applications to Montague Semantics*, North-Holland, 1975.

[26] Girard, J.-Y.: Linear Logic, *Theoretical Computer Science*, Vol.50, pp.1-102, 1987.

[27] Gordon, M. J., Miller, A. J. and Wadsworth, C. P.: Edinburgh LCF, *LNCS*, Vol.78, Springer-Verlag, 1979.

[28] Gordon, M. J. C.: Representing a Logic in the LCF Metalanguage, Neel, D. (ed.), *Tools and Nations for Program Construction*, pp.163-185, Cambridge University Press, 1982.

[29] Gordon, M.J.: HOL – A Proof Generating System for Higher-Order Logic, *VLSI Specification, Verification and Synthesis*, Kluwer Academic Publishers, pp.73-128, 1988.

[30] Griffin, T. G.: An Environment for Formal Systems, ECS-LFCS-87-34, University of Edinburgh, 1987.

[31] Gunter, C. and Gehlot, V.: Nets as Tensor Theories, De Michelins, G. (ed.), *Application and Theory of Petri Nets*, pp.174-191, 1989.

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

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

[34] Hasegawa, R., Fujita, H. and Koshimura, M.: MGTP: A Parallel Theorem-Proving System Based on Model Generation, *Proc. 11th International Conference on Application of Prolog*, 1998.

[35] Hoare, C. A. R.: An Axiomatic Basis for Computer Programming, *Communications of ACM*, Vol.12, No.10, pp.576-580, 583, 1969.

[36] Hopcroft, J. E. and Ullman, J. D.: *Introduction to Automata Theory*, Languages and Computation, Addison-Wesley, 1979.

[37] Hughes, G.E. and Cresswell, M.J.: *An Introduction to Modal Logic*, Methuen, 1968.

[38] ICOT CAP-WG: The CAP Project(1)-(6), *Proc. 32nd Annual Convention IPS. Japan*, 1986. (in Japanese)

[39] Jackson, M.: Evaluation of a Semi-Automated Theorem Prover(Part II), *User Interfaces of Theorem Provers 1997*, 1997.

[40] Jackson, P., et al. (eds.): *Logic-Based Knowledge Representation*, The MIT Press, 1989.

[41] Johnson, S.: Yacc: Yet Another Compiler-Compiler, Computing Science Technical Report, No.32, AT&T Bell Laboratories, 1975.

[42] Jones, C.B., Jones, K.D., Lindsay, P.A. and Moore, R.: *MURAL: A Formal Development Support System*, Springer-Verlag, 1990.

[43] Ketonen, J. and Weening, J. S.: *EKL–An Interactive Proof Checker, User's Reference Manual*, Dept. of Computer Science, Stanford University, 1984.

[44] Kitagawa, T.: The Relativistic Logic of Mutual Specification in Statistics, *Mem. Fac. Sci. Kyushu University*, Ser. A, 17, 1, 1963.

[45] Knowledge Management Server, URL: `http://kman.bus.utexas.edu/kman/`

[46] Kowalski, R.: Algorithm=Logic+Control, *Communications of ACM*, Vol.22, No.7, pp.424-436, 1979.

[47] Kunifuji, S.: New Trends of Knowledge Acquisition and Learning, *Journal of Japanese Society for Artificial Intelligence*, Vol.3, No.6., pp.741-747, 1988. (in Japanese)

[48] Kunst, J.: Making Sense in Music I – The Use of Mathematical Logic, *Interface*, Vol.5, pp.3-68, 1976.

[49] Lakatos, I.: *Proofs and Refutations–The Logic of Mathematical Discovery–*, Worrall, J. and Zabar, E. (eds.), Cambridge University Press, 1976.

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

[51] Lesk, M. E.: Lex – A Lexical Analysis Generator, Computing Science Technical Report, No.39, AT&T Bell Laboratories, 1975.

[52] MacLane, S.: *Categories for the Working Mathematician*, Springer-Verlag, 1971.

[53] Maes, P.: Agents that Reduce Work and Information Overload, *Communications of ACM*, Vol.37, No.7, pp.30-40, 1994.

[54] N. Martí-Oliet and J. Meseguer: From Petri Nets to Linear Logic, *Category Theory and Computer Science*, *LNCS*, Vol.389, Springer-Verlag, pp.313-340, 1989.

[55] Martin-Löf, P.: Intuitionistic Type Theory, *Bibliopolis*, 1980.

[56] McCune, W.W.: *Otter 3.0 Reference Manual and Guide*, ANL-94/6, Argonne National Laboratory, 1994.
URL: `ftp://info.mcs.anl.gov/pub/otter/Papers/otter3_manual.ps.gz`

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

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

[59] 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.

[60] Minami, T. and Sawamura, H.: Proof Constructors for the Reasoning Assistance, *Proc. 33rd Annual Convention IPS. Japan*, 1986. (in Japanese)

[61] Minami, T. and Sawamura, H.: A Construction of Computer-Assisted-Reasoning System, *Proc. 3rd Conference Japan Society for Software Science and Technology*, 1986. (in Japanese)

[62] Minami, T. and Sawamura, H.: Proof Construction with Working Sheets–A Consideration on the Methodology for Computer Assisted Reasoning–, *Proc. 35th Annual Convention IPS. Japan*, 1987. (in Japanese)

[63] Minami, T., Sawamura, H., Satoh, K. and Tsuchiya, K.: EUODHILOS: A General-Purpose Reasoning Assistant System–Concept and Implementation–, IIAS-SIS Research Report, No.84, FUJITSU LTD., 1988.

[64] Minami, T., Sawamura, H., Satoh, K. and Tsuchiya, K.: EUODHILOS: A General-Purpose Reasoning Assistant System–Concept and Implementation–, *LNCS*, Vol.383, Springer-Verlag, pp.172-187, 1989.

[65] Minami, T. and Sawamura, H.: EUODHILOS: An Interactive Reasoning Assistant System, *Journal of Japanese Society for Artificial Intelligence*, Vol.5, No.1, 1990.

[66] Minami, T., Ohashi, K., Sawamura, H. and Ohtani, T.: General-Purpose Reasoning Assistant System EUODHILOS Visual Manual, Research Report IIAS-RR-92-18J, FUJITSU LABORATORIES LTD., 1992. (in Japanese)

[67] Minami, T., Sawamura, H. and Ohtani, T.: A Beginner's Guide to EUODHILOS, Research Report ISIS-RR-94-11E, ISIS, FUJITSU LABORATORIES LTD., 1994.

[68] Minami, T., Ohtani, T. and Sawamura, H.: Reasoning Assistant System EUODHILOS-II, *Fujitsu Scientific and Technical Journal*, Vol.32, No.2, pp.171-182, 1996.

[69] Minami, T., Ohtani, T., Arima, J., and Oda, M.: Agent Society for Network Problem Solving, *Proc. 12th Annual Convention IPS. Japan Kyushu Region*, pp.129-138, 1998. (in Japanese)

[70] Minami, T., Sazuka, H., Hirokawa, S. and Ohtani, T.: Living with ZK–An Approach towards the Communication with Analogue Messages, *Proc. of Second International Conference on Knowledge-Based Intelligent Electronic Systems(KES'98)*, pp.369-374, 1998.

[71] Minami, T., Ohtani, T. and Sawamura, H.: Reasoning Programming, *SIG Programming of IPS. Japan*, 1998. (in Japanese)

[72] Minami, T., Ohtani, T. and Sawamura, H.: Creation and Sharing of Logic-based Domain- and Meta-Knowledge, *Proc. of Pacific Rim Knowledge Acquisition Workshop(PKAW'98) in PRICAI'98*, pp.31-48, 1998.

[73] Minami, T., Ohtani, T., and Sawamura, H.: Usability Evaluation of the General-Purpose Reasoning Assistant System EUODHILOS-II, *International Journal of Knowledge-Based Intelligent Engineering Systems*, Vol.3, No.1, pp.27-36, 1999.

[74] Mizoguchi, R. and Kakusho O.: Knowledge Acquisition Systems, *Journal of Japanese Society for Artificial Intelligence*, Vol.3, No.6, pp.732-740, 1988. (in Japanese)

[75] Mizoguchi, R. and Ikeda, M.: Ontology Engineering–Towards the Basic Theory and Technology for Content-Oriented Research–, *Journal of Japanese Society for Artificial Intelligence*, Vol.12, No.4, pp.559-569, 1997. (in Japanese)

[76] Nonaka, I. and Takeuchi, H.: *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*, Oxford University Press, 1995.

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

[78] Ohtani, T., Sawamura, H. and Minami, T.: Implementing Constructive Type Theory on EUODHILOS, Research Report ISIS-RR-93-14E, FUJITSU LABORATORIES LTD., 1993.

[79] Ohtani, T., Sawamura, H. and Minami T.: EUODHILOS-II on top of GNU Epoch, Bundy, A. (ed.), *Proc. 12th International Conference on Automated Deduction(CADE-12)*, *LNAI*, Vol.814, Springer-Verlag, pp.816-820, 1994.

[80] Ohtani, T., Sawamura, H. and Minami, T.: Reasoning Assistant System EUODHILOS-II: Operation Manual, Research Report ISIS-RR-95-19E, FUJITSU LABORATORIES LTD., 1995.

[81] Ohtani, T., Sawamura, H. and Minami, T.: Design and Implementation of General Reasoning Assistant System EUODHILOS-II, *Trans. IPS. Japan*, Vol.38, No.1, 1997. (in Japanese)

[82] Ohtani, T. and Minami, T.: The Word-of-Mouth Agent System for Finding Useful Web Documents, *Asia Pacific Web Conference(APWeb98)*, National Natural Science Foundation of China, pp.295-300, 1998.
URL: `http://www3.cm.deakin.edu.au/apweb98/FINAL/012.doc`

[83] Parker, J. H.: Social Logics: Their Nature and Uses in Social Research, *Cybemerica*, Vol.25, No.4, pp.287-307, 1982.

[84] Paulson, L. C.: The Foundation of a Generic Theorem Prover, *Journal of Automated Reasoning*, Vol.5, pp.363-397, 1989.

[85] Paulson, L.C.: Isabelle: A Generic Theorem Prover, *LNCS*, Vol.828, Springer-Verlag, 1994.

[86] Peirce, C. S.: *Collected Papers of C. S. Peirce*, Hartshorne, Ch. and Weiss, P. (eds.), Harvard University Press, 1974.

[87] Pereira, F. C. N. and Warren, D. H. D.: Definite Clause Grammars for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artificial Intelligence*, Vol.13, pp.231-278, 1980.

[88] Peyton Jones, S. L.: *The Implementation of Functional Programming Languages*, Prentice-Hall, 1987.

[89] Prawitz, D.: *Natural Deduction – A Proof-Theoretical Study*, Acta Universitatis Stockholmiensis, Stockholm Studies in Philosophy, 3rd ed., Stockholm, Almqvist & Wiksell, 1965.

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

[91] Reps, T.: *The Synthesizer Generator Reference Manual*, Department of Computer Science, Cornell University, 1985.

[92] Resnick, P. and Varian, H. R. (Eds.): Recommender Systems, *Special Issue of Communications of ACM*, Vol.40, No.3, 1997.

[93] Ritchie, B. and Taylor, P.: The Interactive Proof Editor–An Experiment in Interactive Theorem, ECS-LFCS-88-61, University of Edinburgh, 1988.

[94] Sakai, K.: CAP: Computer Aided Proof, *Journal of Japanese Society for Artificial Intelligence*, Vol.5, No.1, pp.33-40, 1990. (in Japanese)

[95] Satoh, K., Tsuchiya, K., Ono, E., Sawamura, H. and Minami, T.: Well-Formed Formulas Editor for Argumentation Supporting System, *Proc. 33rd Annual Convention IPS. Japan*, 1986. (in Japanese)

[96] Sawamura, H.: A Proof Constructor for Intensional Logic, with S5 Decision Procedure, IIAS Research Report, No.65, FUJITSU LTD., 1986.

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

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

[99] Sawamura, H., Minami, T., Yokota, K. and Ohashi, K.: Potential of General-Purpose Reasoning Assistant System EUODHILOS, Nakata, I. and Hagiya, M. (eds.), *Software Science and Engineering: Selected Papers from the Kyoto Symposia*, World Scientific, 1988, Also Research Report IIAS-RR-91-8E, FUJITSU LABORATORIES LTD., 1991.

[100] Sawamura, H., Minami, T., Ohtani, T., Yokota, K. and Ohashi, K.: A Collection of Logical Systems and Proofs Implemented in EUODHILOS I, Research Report IIAS-RR-91-13E, FUJITSU LABORATORIES LTD., 1991.

[101] Sawamura, H., Minami, T. and Ohashi, K.: EUODHILOS: A General Reasoning System for a Variety of Logics, Voronkov, A. (ed.), *Proc. of International Conference on Logic Programming and Automated Reasoning*, *LNAI*, Vol.624, Springer-Verlag, pp.501-503, 1992.

[102] Sawamura, H., Minami, T. and Ohashi, K.: Proof Methods based on Sheet of Thought in EUODHILOS, Research Report IIAS-RR-92-6E, FUJITSU LABORATORIES LTD., 1992.

[103] Sawamura, H., Minami, T. and Ohtani, T.: Application and Evaluation of General-Purpose Reasoning Assistant System EUODHILOS, *Trans. IPS. Japan*, Vol.34, No.5, pp.809-819, 1993, Also Research Report IIAS-RR-93-2J, FUJITSU LABORATORIES LTD., 1993. (in Japanese)

[104] Sawamura, H., Minami, T., Yokota, K. and Ohashi, K.: A General-Purpose Reasoning Assistant System EUODHILOS–Basic Features and Potential Usefulness–, *Trans. IPS. Japan*, Vol.36, No.3, pp.542-560, 1995.

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

[106] Slaney, J. and Meglicki, G.: MaGIC: Matrix Generator for Implication Connectives, INTERIM VERSION 2.0 NOTES AND GUIDE, Technical Report TR-ARP-1/91, Australian National University, 1991.

[107] Slaney, J.: FINDER: Finite Domain Enumerator, Technical Report TR-ARP-1/92, Australian National University, 1992.

[108] Smullyan, R.: *To Mock a Mockingbird*, Alfred A. Knopf Inc., 1985.

[109] Stallman, R.M.: *GNU Emacs Manual*, Free Software Foundation, 1994.

[110] Thistlewaite, P. B., McRobbie, M. A. and Meyer, R. K.: *Automated Theorem-Proving in Non-Classical Logics*, Pitman Publishing, 1988.

[111] Trybulec, A. and Blair, H.: Computer Assisted Reasoning with MIZAR, *Proc. of IJCAI'85*, pp.26-28, 1985.

[112] Tsuchiya, K., Satoh, K., Ono, E., Sawamura, H. and Minami, T.: Well-Formed Formulas Editor for the Reasoning Assistant System, *Proc. 35th Annual Convention IPS. Japan*, 1987. (in Japanese)

[113] Turner, A.: *Logics for Artificial Intelligence*, Ellis Horwood Limited, 1984.

[114] Weyhrauch, R. W.: Prolegomena to a Theory of Mechanized Formal Reasoning, *Artificial Intelligence*, Vol.13, pp.133-179, 1980.

[115] Winston, P. H. and Horn, B. K. P.: *LISP*, Addison-Wesley, 1981.

[116] Zanardo, A. and Rizzotti, M.: Axiomatization of Genetics 2, Formal Development, *Journal of Theoretical Biology*, Vol.118, pp.145-152, 1986.