ICOT Technical Memorandum: TM-0812

TM-0812

対話型論証支援システムEUODHILOS

南　俊朗，沢村 一(ICOT)

October, 1989

ICOT
Mita Kokusai Bldg. 21F
4 28 Mita 1-Chome
Minato ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

対話型論証支援システム EUODHILOS

南 俊朗，沢村 一

富士通 (株) 国際情報社会科学研究所
410-03 沼津市宮本 *140*
E-mail: minami@iias.fujitsu.co.jp

1 はじめに

論証 (Reasoning) は数学，計算機科学，人工知能等多くの分野で重要な役割を果たしている [19, 10]. それらの分野では一階，高階，等号，様相，直観，型理論等，様々な論理が用いられるのみならず必要に応じて新しい論理が生み出される. 本稿では論証を，人間がその思考を何らかの形で形式化し，その特徴を客観的に捉えたり，形式化された思考の方式（すなわち，論理）を用いて様々な結果を得ることと定義する. このように捉えると，論証はごく限られた一部の人達のみが行うものではなく，多数の人々を対象とする活動であることになる. 我々は様々な論理によるこのような論証活動に対して計算機によって特定の論理に依存しない方式での支援を行うシステムの研究を行ってきた [14].

論証と計算機の関わりを考えたとき，まず思い当たるのは定理の自動証明である. 自動証明の研究は歴史も古く多くの成果が得られている [2]. しかし，特定の分野のある限られた範囲の定理証明ではなく，もっと一般的な論理系，定理を対象として考えると自動証明の適用範囲はまだまだ限られていると言わざるを得ない. この限界を越える方法として，決められた仕事を正確に実行する機械と全体状況を見通すことを得意とする人間とが，それぞれの特徴を出し合い補い合いながら協調して論証を進める対話型の論証システムの形態が考えられる. 対話型の論証システムにおいては，人間の側は全体の状況を把握し論証をどのように進めるかを決定しなければならない. そのためにはシステムはユーザが状況を正確に把握するために必要な情報を提示しなければならないし，また，人間にとって使い勝手の良いユーザ・インタフェースを備えていることが強く望まれることになる.

EUODHILOS はこのような汎用の論証支援システムに関する実験を行うために開発されたシステムである. 我々はこのシステムによる論証経験を基に理想的な論証支援システムの形態を明らかにすることを目指している. システムの名前は S. K. Langer[12] の次の言葉に由来している：

"Every universe of discourse has its logical structure."

これは，全ての議論領域はその論理構造を持つという考えを表わしている. すなわち我々が考慮する各の対象領域に対してそれを表現し，それについて議論するための最適の論理があるものであると言う考えである. この考え方を実践するために EUODHILOS は論理系定義機能を備えておりユーザが与えた論理系に基づいた論証を支援する. 汎用の論証支援システム EUODHILOS を用いることで，論理的表現を用いた様々な種類の知識を統一的な枠組で取り扱うことが可能となる.

もう1つの重要な問題は論証に適したユーザ・インタフェースの探求である. 我々は, 論証と言うものは本来試行錯誤的に進むものだということを基本的認識とする. 論理系を記述するにしても, 何らかの論理式に証明を与えるにしても, 一度で最終的なものを得ることは極めて困難である. 従って, 一旦得られた結果を変更したり消去したりが容易に行えなければならない. 証明活動に対して, EUODHILOS は思考シートと呼ばれる定理およびその証明を構築するための編集支援環境を提供している. そこでは, 仮定を置いたり, 上向き, 下向きの導出を自由に混ぜて行ったり, 証明を結合, 分離するなどの証明操作が自由に行える.

以下, 2節で EUODHILOS の概要を述べ, 3節で他の形態の論証システムとの比較を通じて EUODHILOS の特徴を明らかにする. 最後の4節でこれまでの使用経験から得られた知見および将来の課題について整理する.

2 EUODHILOS の概要

前節で述べた特徴を備えた論証支援システムを実現するために EUODHILOS を設計するに当たって, 次の2点が考慮された:

(i) システムの汎用性の実現法

(ii) 論証に適したインタフェースの在り方

その結果, 汎用の記法による構文定義に基づく言語系並びに2種類の導出規則を持つ論理系定義の支援機能, 及びマルチウィンドウによる証明支援のための思考シートを持った EUODHILOS が開発された.

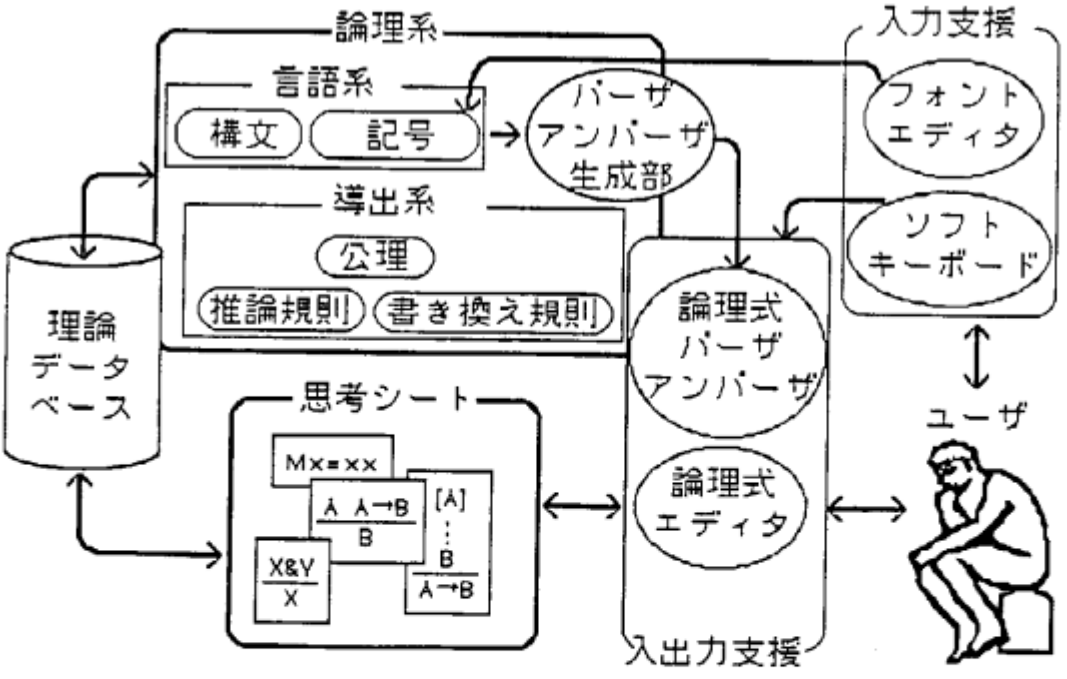

図 1: EUODHILOS の構成

図 1 は EUODHILOS の構成を表わしたものである. 図の上の方の論理系と書かれた枠内が主に汎用性に対応している. ユーザは構文と記号からなる言語系, 及び, 公理, 推論規則, 書き

換え規則からなる導出系を与えることで対象とする論理系を記述する. システムのパーサ・アンパーサ生成部は言語系の記述より，文字列表現を内部表現に変換するパーサと，その逆方向の変換を行うアン・パーサを同時に自動生成する. 下の方にある思考シートは，主に特徴 (ii) に関連しており，ユーザは前段階で定義した論理の下での定理の証明をその支援の下で行う. 思考シート上に完成した証明結果は理論データベースに定理として保存でき，それは別の定理の証明のために再利用できる. このようにして，ユーザはその理論の中で多くの定理を累積し利用できる. また，論理表現の入出力を支援するためにフォントエディタ，ソフトウェアキーボード，そして論理式エディタが備えられている.

2.1 言語の記述

EUODHILOS においては，論証の基礎となる言語系はユーザが最初に設計し，システムに与える. 言語系は，文字列としての論理表現の組み立てを確定節文法 (DCG)[16] を用いて与え，それが内部で解釈されるときにどの要素が中心要素と見做されるかを演算子定義によって指定する [15] ことで定義する. 与えられた言語系より，定義された論理表現に対する（ボトム・アップ）パーサ [13] とアンパーサが自動的に生成される. パーサは，文字列である外部表現より，内部表現へ変換を行い，アンパーサはその逆変換を行う. パーサとアンパーサはその後の記号操作の全ての段階において外部表現と内部表現の間の変換が必要な場合に自動的に呼ばれ用いられる. 文字列の表現が入力されると，パーサが呼び出されその正当性がチェックされる. それと同時に，その言語における，その表現の内部構造が生成される. 従って EUODHILOS の構文定義において表現の内部構造を明示的に指定する必要なない. 導出コマンドがユーザにより与えられた時，論理式の内部表現が操作されあらたな内部表現が生成される. 生成された内部表現はアンパーサによって外部表現に変換された後，ユーザに提示される.

図 2 は R. Smullyan[18] による物真似鳥 (Mocking bird) のパズルに対する言語記述の例である. これは組み合わせ論理に対して様々な鳥の住む森の世界への解釈を与えらものである. 図の定義により，“M•x=x•x” とか “(A*B)•x=A•(B•x)” とかの表現がその論理における式（formula）であることがわかる. メタ記号は公理の定義，推論規則や書き換え規則，そして，思考シート上の図式（スキーマ）による証明で用いられる.

2.2 公理及び導出規則の記述

EUODHILOS における導出系は公理と導出規則からなる. 導出規則としては推論規則と書き換え規則が用いられる. 公理として論理式のリストが与えられる. 推論規則は自然演繹法スタイルで与えられる. すなわち，推論規則は 3 つの部分からなる. 1 つめは規則の前提であり，それぞれの前提は仮定を持つことができる. 2 つめは規則の結論であり，最後に 3 つめは変数の出現等適用に関する条件である. 図式的に表わすと，推論規則は次の形式で与えられる（条件部分を除く）:

$$\frac{\begin{array}{cccc}[\text{仮定}_1] & [\text{仮定}_2] & \cdots & [\text{仮定}_n] \\ \vdots & \vdots & & \vdots \\ \text{前提}_1 & \text{前提}_2 & \cdots & \text{前提}_n\end{array}}{\text{結論}}$$

この形式において，仮定の部分は任意である. ある前提がその仮定を持つならば，それはその前提がその仮定の下で得られることを示し，そうでない時は，何らかの方法でその前提が得られることを示している. 推論規則はその適用に関して条件を持つことができる. 推論規則は，その全

```
formula → term, "=", term;
term → b_term;
term → b_term, "•", b_term;
b_term → variable_symbol;
b_term → constant_symbol;
b_term → b_term, "*", b_term;
b_term → "(", term, ")";

variable_symbol → "A"|"B"|"x";
constant_symbol → "M";

meta_formula → "F";
meta_formula → meta_pred, "[", term, "]";
meta_pred → meta_formula;
meta_variable → "X";
meta_term → "Y"|"Z";
b_term → meta_term.

operator
  "*";   "•";   "=";
predicate
  meta_pred.
```

図 2: 物真似鳥パズルに対する言語記述

ての前提がこのように得られ，かつ制限条件が（もし指定されている時）満足されたならば適用可能である．

図 3 は物真似鳥の論理に対する公理と推論規則である．'sbst' と名付けられた推論規則の定義において，'[X]' や '[Y]' の部分はそれぞれ，論理式 'F' における変数 (variable) 'X' や項 (term) 'Y' の出現を表わしている．

公理：

M•x=x•x　物真似鳥が存在する

(A*B)•x=A•(B•x)　鳥の合成

推論規則：

$\frac{F[X]}{F[Y]}$ (sbst 代入)　　$\frac{F[Y] \quad Y=Z}{F[Z]}$ (eq 等値)

図 3: 物真似鳥の論理の公理及び推論規則

論証の典型例である数学分野における論証過程を観察して見ると，式の書き換えによる推論が多用されていることに気づく．この認識を踏まえ EUODHILOS は導出規則の 1 つとして書き換え規則を許している．書き換え規則は次の形式で与えられる：

$$\frac{\text{書き換え前表現}}{\text{書き換え後表現}}$$

書き換え規則がある表現に適用されるのは，それがその規則の“書き換え前表現”部分に合う部分表現を持っている場合である．結果の表現はその部分表現を規則の対応する“書き換え後表現”に置き換えて得られる．

導出規則を繰り返し適用することで導出木を得ることができる．多くの場所に現れている幾つかの規則の類似の適用を簡約化するために派生規則を定義することができる．一旦定義されると，派生規則は基本規則と同様に用いることができる．

Hilbert 流，Gentzen 流，等号などの記法がこの枠組の中で記述できる．たとえば，Hilbert 流の記述は公理を主とし，推論規則はモダスポネンスと一般化のみとすることで可能である．Gentzen 流はそれと対照的に，公理はなく，推論規則のみとなる．

2.3 証明の構成

EUODHILOS においては思考シートと言う名の定理とその証明の発見の支援環境が用意されている．それは，証明の草案，証明断片（すなわち，部分的に構成された証明）の結合，分離，補題による論証等を許す．それは，定理や証明は試行錯誤的にユーザの主導権の下で行われると言う我々の考えに基づいて設計されている．

思考シート上では，証明断片が処理の基本単位である．証明断片は，仮定，公理，以前に得られた定理として新しく生成される．推論及び書き換え規則はちょうど紙の上に印刷されているのと同様に図式的に表現され適用される．その結果，シート上の導出木の外観も木の形式で表示されることになる．このように証明が視覚的に行え，自然な論証形態の支援が考慮されている．

証明を構成することも人間の自然な論証形態に沿って行われることが望まれる．EUODHILOS は従来のシステムとは異なり，様々な論証形態を支援する．すなわち，前向き（トップ・ダウン），後向き（ボトム・アップ）の導出が可能であり，更にはそれらを自由に混合することも可能である．ユーザは仮定式を思考シート上に置いたり，前向きや後向きの導出によって，より大きな証明断片を作り上げるたり，また，既に得られた証明断片を結合したり，逆に 1 つの証明断片を分離したりを自由に組み合わせて目的とする定理の証明を構築を行える．また，証明断片はメタ変数を用いた式を含むことができるため，初めは一般的表現で論証を進め，必要になった時点でメタ変数を具体化することができる．このため，初めから後で必要となるはずの式の形を予測してその式についての導出を行わなければならないといったユーザにとって負荷の大きい要求が不要となる．

思考シート上の導出過程がどのようなものであるかをより深く理解するために，物真似鳥の例において導出がどのように行われるかの一例を示そう．問題は，次の文を証明することである：

“どの鳥も，ある鳥から好かれる．”

これは，任意の鳥 ‘A’ に対して，“A•X=X” であるような鳥 ‘X’ が存在すると言う意味である．図 4 はこの例に対する思考シートの実画面である．まず最初に，図の右上の角にある思考シートに見られるようにユーザが 2 つの公理 “M•x=x•x” と “(A*B)•x=A•(B•x)” を入力したとしよう．これらの式から何らかの導出を行うために，変数 ‘x’ に ‘A’ を代入し，“M•x=x•x” という公理から “M•A=A•A” を導出したとする．この場合，これ以上導出を進めることができない．それで，他のやり方を試みることにする．

左中ほどにある思考シートに見られるように，次に ‘x’ に ‘A*B’ を代入してみよう．今回は，“M•(A*B)=(A*B)•(A*B)” 及び “(A*B)•(A*B)=A•(B•(A*B))” が得られる．これらを眺めて見ると，‘M’ と ‘B’ に代入することで，目的の式である “(A*M)•(A*M)=A•((A*M)•(A*M))” が得られ

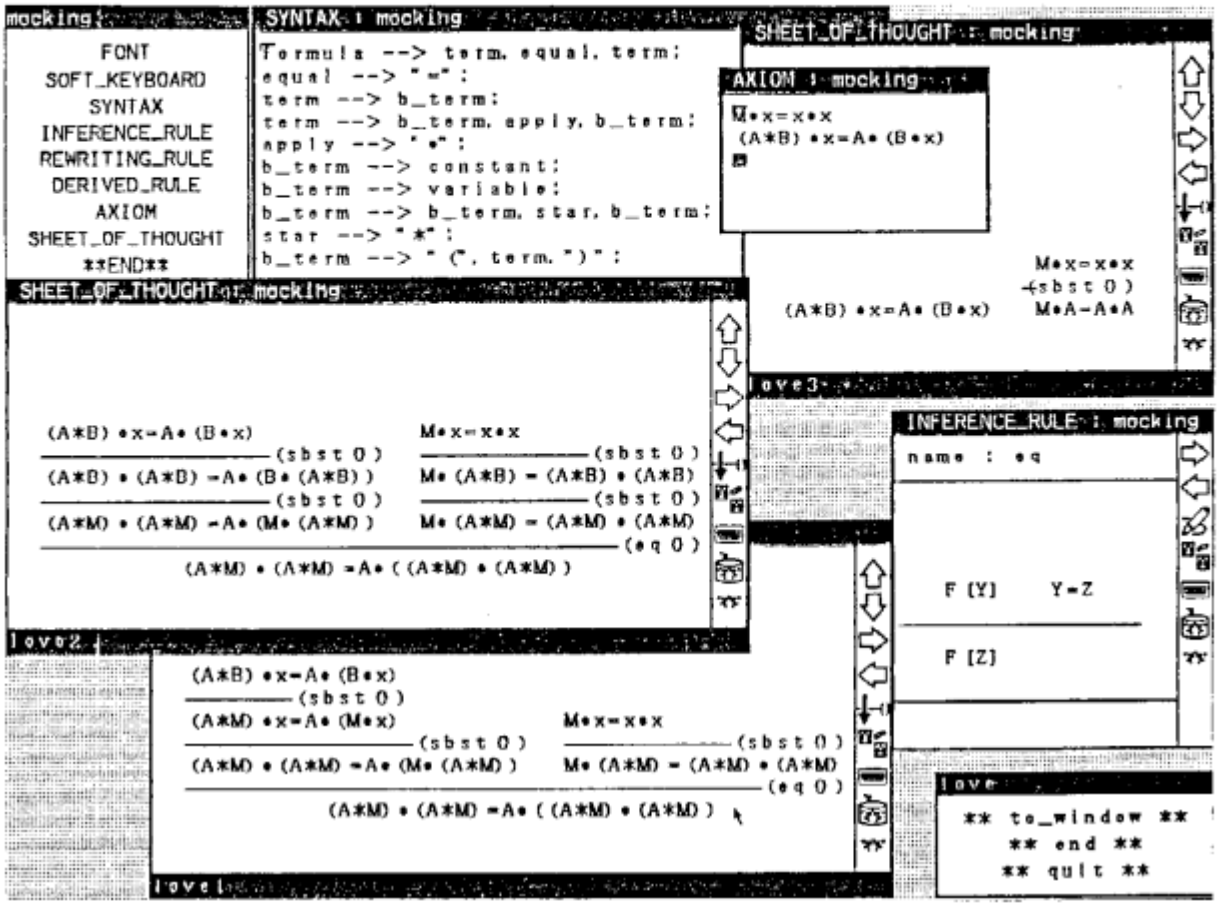

図 4: 思考シート上の証明構成

ることに気付く. これは, 鳥 'A' は "(A*M)•(A*M)" という表現で表わされる鳥によって好かれると言うことを表わしている.

ここで, その証明を注意深く眺め直して見ると, 証明に無駄があることに気付く. その結果, 定理の最終的証明が得られる. 'A*M' を 'x' に, そして 'M' を 'B' に代入し, 等号に対する推論規則を用いることで目的の式が得られる. (それは, 左下のシートに現れている.) 思考シート上の証明はこのよう過程を経ながら進んで行く. もっと, 実際的な例については [14] 及びその中の他の論文を参照されたい.

3 他の論証システムとの比較

本節では, 論証支援システム (RAS と略す) を人間の行う論証を支援するために用いることのできる次のような 3 つの形態のシステムと比較する :

(1) 自動定理証明機 (Automated Theorem Prover–ATP),

(2) 証明チェッカ (Proof Checker)

(3) 証明構成機 (Proof Constructor)

ATP はユーザによって与えられた論理式の証明を探索するシステムである. 一方 RAS においては, システムとユーザの対話を通じて証明が探索され発見される. 主導権はユーザに有る. こ

れはATPとRASの主な相違点である. この違いは結局その目的とする点の違いに帰着できよう. ATPの研究は, その理論的興味による部分も大きいと思われるが, 人間の行う論証との関わりという観点で考える時, 人間の行う論証と言う知的活動を機械的にどのように実現するかというのが最大の目標であるように思われる. それに反してRASの目標は, 知的活動を機械的に実現することを一旦諦め, 人間の足りない部分を機械でうまく補おうということである. そのようなシステムにおいて重要なのは人間と機械全体として厳密でかつ効果的な論証をいかにして行うかということである. 目標という点では対照的であるが, これら2つの論証形態は実は相補的である. 全てを自動的にというのは実際的ではないし, 一方ATP的な機能をもたないRASにおいては, 導出ステップを踏まないと定理を証明出来ないことになり, 手間が掛かる. 結局, RASの立場から考えた理想の姿はその基本的形態はRASであるが, システムを知的に高度化するのにATPの成果を有効に利用したものとなる.

証明チェッカはユーザによって記述された証明の正しさを検証するシステムである. 証明チェッカにおいて, ユーザは最初に定理の証明とおぼしきものを持っている. 人間の証明は証明の中の小さなギャップを含めた不注意による誤りを含んでいる可能性が有る. 証明チェッカはこの人間の証明を記述する言語を提供する. ユーザはその証明をこの言語で記述しそれをチェッカに与える. チェッカはこの証明を妥当性をチェックする. 証明を計算機に入力するためには通常用いられている, テキストエディタを用いることになる. このテキストをシステムに渡すことにより, システムはバッチ的に処理し, その結果を返す. ユーザが証明を既に得ており, その正しさを検証したい場合には証明チェッカは最良の道具である. しかし, ある論理式の証明を発見しようと言う時は証明の構成を支援するシステムの方がより良い支援を与えられる.

多くの証明チェッカがこれまで開発されている. AUTOMATH [1] はユーザが証明の構成を規定するシステムである. PL/CV2 [3] は PL/I と似たプログラムの正しさを証明するために用いられる. CAP-LA [9] は線形代数の証明を扱う.

証明コンストラクタ (e.g. LCF [6], FOL [20], EKL [11], IPE [17], Nuprl [4]) はユーザとシステムの対話によって定理と同様に証明を構成することを支援する. 証明チェッカとの違いは, 証明チェッカのユーザはシステムに証明のチェックを頼む前に人手で得られた証明を持っていると言う状況を想定していたのに対し, 証明コンストラクタは, 証明の構成過程そのものを支援し, 始めから正当性の保証された証明を作り上げようという状況を想定している点に有る.

この点において, RASは証明コンストラクタと見ることができるかも知れない. しかし, 重要な違いがある. それはRASが汎用であることである. 汎用であることより, 様々な論理において行われている人間の論証の全てが適用の対象となる. 論理系を固定している一般の証明コンストラクタの場合は, その適用領域を固定して (あるいは限定して) 考えていることになる. 論理を固定することで考慮の対象となっている論証を制限しているわけである. そうすることで, その対象領域特有の性質を利用した特殊な手続きをシステムに組み込むことが容易であり, 実際そのような機能と結合して使われている場合が多いように思われる. RASのアプローチの場合は, その扱う論理系を固定していないため, 特殊な論理系そのものを試行錯誤的に作り上げることにも利用可能である. したがって, 記述したい領域が特殊な論理的構造をもっていても直ちにその上の論証実験を行うことが出来る. 論理系に依存した特殊な処理の導入は証明コンストラクタと比べると若干手間が掛かるかも知れない. しかし, 特殊処理を扱うための手続きを加える枠組を実現することで, このような処理機能に対しても原理的に対処可能である.

EUODHILOSにおける証明断片の表現はそれ以外の既存の証明コンストラクタとのもう1つの大きな相違である. そこでは, 自然な論理式や証明木の表現が用いられている. 従って論理式を読んだり, どのように証明を行おうとしているかの状況の把握が容易である. そして, 一見僅

かなものに思われるこの長所は，困難なそして複雑な証明を構成する際には大きな相違をもたらす．結局，EUODHILOSのユーザ・インタフェースを設計する際に考慮したことは，日頃人間が紙や黒板の上に表現している論理表現を可能とし，またそこで行っている論証の形態をいかにしてディスプレイの上に実現するかということであったと言うことができよう．この意味での自然さは将来においても論証に適した環境を実現するための重要なポイントであろう．

4 むすび

現在動いているEUODHILOSは論証の実験版であり，その使用経験をより高度な論証支援機能の設計に活かしたいと考えている．

これまで，一階論理(NK)，命題様相論理(T)，内包論理(IL)，組み合わせ論理，Martin-Löfの型理論，そして圏論といった多くの重要な論理が実際に現在EUODHILOSで記述された．tableau法などの幾つかの論理は現在の枠組では取り扱うことができないが，枠組を拡張することでこれらも扱えることを目指したい．

EUODHILOSにおけるこれまでの経験より，我々は次の知見を得た:

(i) 経験の乏しい最初の頃は，論理表現の構文を記述することは困難であり，様々な誤りを経験した．しかし，幾つかの論理を定義した後では，新しい論理でも数時間で定義できるようになることが分かった．システム内に典型的な論理記述をライブラリとして保存することで，初心者でも新しい論理を容易に記述できるものと考えられる．

(ii) 思考シート上で，ユーザは導出エラーから開放される．紙の上では，導出規則を適用する時新しい論理式を導くのに誤りを犯すかも知れない．この違いは重要である．と言うのはユーザは思考シート上では証明をどのように進めるかの決定にのみ注意を払えば良いからである．

(iii) 論証支援システムはCAIの道具として用いることができる．システムの中で，ユーザは多くの論理を扱うことができる．この用途にRASを用いるには段階的な証明の自動化機能が有効である．初心者に対しては余り自動化されない形態の支援を行い，手間を掛けて経験することが論証のスキルを身につけるためには役立つものと思う．学習者の段階の応じて自動化の度合いを増やして行けば，その段階に応じた論証の学習支援システムになるものと考えられる．

現在の状態は実際的な論証支援システムを実現するための第一ステップである．このステップを更に進めるためには，次のテーマを含む様々な問題を探求する必要がある：

(1) メタと対象の理論の間の関係の取り扱い．

(2) 様々な理論間の依存関係の管理．

(3) 論証のための新しい適用領域の開発．

(4) 論証システムのための人間 - 計算機間のインタフェースの改良．

謝辞

我々の共同研究者である富士通研究所の横田かおる，大橋恭子の両氏に感謝する．本研究は第五世代コンピュータ・プロジェクトの一環としてICOTの委託で行ったものである．

参考文献

[1] N.G. de Bruijn: The Mathematical Language AUTOMATH, its Usage, and some of its Extensions, In M. Laudet et al. (eds.), *Sympoosium on Automated Demonstration*, Springer-Verlag, pp.29–61, 1970.

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

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

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

[5] J.A. Goguen & R.M. Burstall: Introducing Institutions, *LNCS* **164**, Springer-Verlag, 1983.

[6] M.J. Gordon et al.: Edinburgh LCF,*LNCS* **78**, Springer-Verlag, pp.221–270, 1979.

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

[8] R. Harper, F. Honsell & G. Plotkin: A Framework for Defining Logics, *ECS-LFCS-87-23*, Univ. of Edinburgh, 1987.

[9] ICOT: The CAP Project (1)–(6), 情報処理学会第３２回全国大会, 1986.

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

[11] J. Ketonen & J.S. Weening: EKL—An Interactive Proof Checker, User's Reference Manual, *Dept. of Computer Science*, Stanford Univ., 1984.

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

[13] Y. Matsumoto et al.: BUP:A Bottom-Up Parser Embedded in Prolog, *New Generation Computing* **1**, pp.145–158, 1983.

[14] T. Minami et al.: EUODHILOS: A General-Purpose Reasoning Assistant System – Concept and Implementation –, IIAS-SIS Research Report No. 84, *FUJITSU LIMITED*, 1988.

[15] 大橋他：確定節文法のための内部構造変換機能付きパーザとアンパーザの自動生成方式, 投稿準備中, 1989.

[16] F.C.N. Pereira et al.: Definite Clause Grammars for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks, *AI Journal* **13**, pp.231–278, 1980.

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

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

[19] R. Turner: Logics for Artificial Intelligence, Ellis Horwood Limited, 1984.

[20] R.W. Weyhrauch: Prolegomena to a Theory of Mechanized Formal Reasoning, *AI Journal* **13**, pp.133-179, 1980.