Vol. 38 No. 1 情報処理学会論文誌 Jan. 1997

汎用論証支援システムEUODHILOS-IIの設計と実装

大谷 武[†] 沢村 一[††] 南 俊朗[†]

EUODHILOS-IIは，PSI/SIMPOS上に実現されたEUODHILOSの可搬性と利用可能性を高めるために，GNU Emacs上に新たに作られた汎用の論証支援システムである．この論文では，まずEUODHILOS-IIを広く普及しているGNU Emacs上に設計実現する際に問題となった諸点を議論する．次いで，EUODHILOS-IIの主要な構成要素である論理系定義，証明構築，証明指向のインタフェースに含まれる特徴的な仕様設計と実現方法について述べる．最後に，EUODHILOS-IIの利点をまとめ，設計思想が類似している他のシステムとの比較を与える．

Design and Implementation of General Reasoning Assistant System EUODHILOS-II

TAKESHI OHTANI,[†] HAJIME SAWAMURA[††] and TOSHIRO MINAMI[†]

EUODHILOS-II is a general reasoning assistant system for various logics built on top of GNU Emacs, aiming at highly portable and widely usable version of EUODHILOS built on PSI/SIMPOS. This paper first argues about the issues which raised in designing and implementing EUODHILOS-II on the widely prevailing platform GNU Emacs. Then, we describe the specification and the implementation methods of unique features in logic definition, proof construction and reasoning-oriented interface which are the main components of EUODHILOS-II. The paper concludes with discussing some advantages of EUODHILOS-II and comparing it with other systems which have the similar design principle.

1. はじめに

計算機科学や人工知能など様々な分野で，論理を利用した問題解決方法の有効性が認識され，数多くの論理系が提唱されてきた．それと同時に，それらの論理系における証明構築を計算機によって行う手法も研究されてきた．多くの論理系においては決定可能性が成立しないために，すべての証明が計算機によって完全に自動的に行われる自動証明機は望むべくもない．そこで，我々は人間が持つ発想と計算機が持つ強力な記号操作と探索機能の協調によって効率良く証明を行うことのできる対話型証明構築支援システムを目指し，汎用の証明構築支援システムEUODHILOS[6),11),12)]を開発した．しかし，動作する計算機がICOTで開発された逐次型推論マシンPSIに限定されていたため利用者も限定され，得られた証明データを別の形で利用することも困難であった．また，EUODHILOSの使用経験より，いくつかの改善点が指摘された．EUODHILOS-II[7),8)]は，EUODHILOSの論証支援スタイルを受け継ぎつつも，これらの問題点を解決し，論理系の記述法や証明支援機能の強化や操作性の改良と広範囲の計算機上で動作することを目指して，新たに設計・実装された．

EUODHILOS-IIは，図1で示されるように，各論理系ごとに論理系で使用する言語および公理系を定義する論理系定義部と実際にユーザが証明を構築する証明構築支援部から構成される．ユーザは，まず論理系定義部の一部である言語系定義部で自分の思い描いている論理系で使用する言語を定め，その言語を用いて，公理・推論規則・書き換え規則からなる導出系を導出系定義部で規定する．次に，この論理系定義を基に証明エディタを使用して証明を行い，目標とする問題の解決を図る．証明によって得られた定理や派生規則は理論データベースに蓄えられ，公理や推論規則と同じように他の証明の中で使用することができる．証明を行う際に論理系の不備が見つかるか，あるいは問題自身の修正が必要であれば，随時言語系や導出系の

† 株式会社富士通研究所パーソナルシステム研究所
Personal Systems Lab., Fujitsu Lab. Ltd.
†† 新潟大学工学部情報工学科
Department of Information Engineering, Niigata University

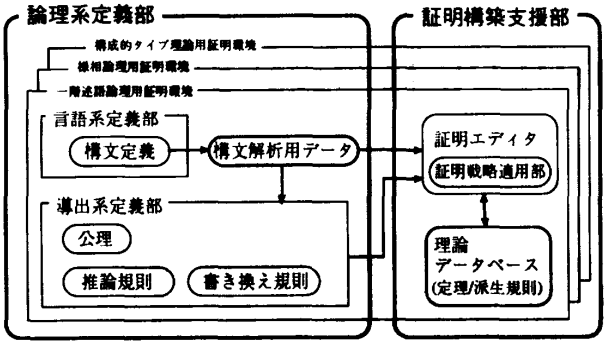


図1 EUODHILOS-IIのシステム構成
Fig. 1 System configuration of EUODHILOS-II.

定義を修正し，再び証明を続けることができる．また，EUODHILOS-IIには，EUODHILOSでは実装されていない証明戦略による自動証明法が組み込まれており，類似の証明手順の繰返しや，見通しの良い証明を1ステップで行うことで，ユーザの手間をより一層軽減することができる．

本論文の目的は，EUODHILOS-IIの特徴や機能，およびそれらの設計思想や実装法を論じることによって，EUODHILOS-IIの汎用論証支援システムとしての有効性を明らかにすることにある．以下，2章では，EUODHILOS-IIをGNU Emacs[13]上に実装することの利点と有効性を論じ，3章と4章ではそれぞれ論理系定義と証明構築に関する機能の主要な部分を抽出し，その仕様設計と実現法を具体的に示す．そして，5章では，EUODHILOS-IIのユーザインタフェースの代表的な特徴について述べる．最後に，6章で他のシステムとの比較をし，結論を述べ，今後の課題として取り組むべき重要な機能を指摘する．

2. 実装のためのプラットフォーム

我々は以下に述べる観点を考慮した結果，EUODHILOS-IIを実現するプラットフォームとしてGNU Emacs[13]を採用し，その機能拡張言語Emacs Lispを用いてEUODHILOS-IIを実装した．

- **マルチウィンドウ環境**：EUODHILOS-IIにおいては，論理系の定義と構築途中の証明を交互に見比べる必要が生じる．また，大規模な証明を構築するために，いくつかの証明断片を作成し，同時に表示させ，それらを組み合わせて証明を完成させるのは有効な方法である．したがって，容易にマルチウィンドウシステムを構築できることが望ましい．
- **操作の一貫性**：ユーザにとって普段使い慣れている操作法で新しいシステムを利用できることは大きな利点である．GNU Emacs上には様々なアプリケーションが実現されており，多くのユーザが日常的にそれらを利用している．そのため，同じ操作性を実現することで，EUODHILOS-IIの操作性を修得しやすくすることができる．また，本来GNU Emacsが持っている編集機能をEUODHILOS-IIの中で利用することも可能である．
- **可搬性**：EUODHILOS-IIの可搬性やデータの流通性を確保するためには，多くの計算機あるいはOSで稼働しているプラットフォームを選ぶ必要がある．GNU Emacsはすでに多様な環境で稼働しているという実績があり，その機能拡張言語Emacs Lispのみを用いて実装することで，EUODHILOS-IIをGNU Emacsと同様に多くの計算機で稼働させることができる．
- **カスタマイズと拡張性**：EUODHILOS-IIのような対話的なシステムにおいては，ユーザの好みを反映させるカスタマイズ機能が必要であるが，GNU EmacsではEmacs Lispを使用して容易にカスタマイズを行うことができる．また，ユーザ自らEmacs Lispでプログラムを作成し組み込むことにより，EUODHILOS-IIの機能拡張が容易である．
- **環境の局所化**：EUODHILOS-IIのような汎用の証明エディタでは，同時に複数のアプリケーションを起動し，論理系定義や証明を編集できる機能が必要である．また，同時に複数の論理系を操作できる機能が必要となる場合もある．これはEmacs Lispのバッファローカル変数を利用して容易に実現することができる．この変数はバッファごとに局所化された変数であり，各バッファで異なる値を保持することが可能である．このような特性はオブジェクト指向のカプセル化という概念に類似している．

3. 論理系定義法の実装

EUODHILOS-IIは，ユーザが容易に論理系定義を行うことができ，一階述語論理やタイプ理論などのよく知られた論理系を自然に表現できる記述法として，言語系定義には文脈自由文法を，導出系定義には自然演繹法[10]の枠組みを用いた．

3.1 言語系定義の記述法

EUODHILOS-IIの構文定義は，**図2**の実例に示されるように，比較的平易な文脈自由文法に基づき，*ROOT*, *META_VARIABLES*, *PRODUCTIONS*の3つの部分からなる．*ROOT*は，論理式など証明で対象となる文字列に付与される非終端記号を開始記号として宣言する．*META_VARIABLES*では，自由に具体化が可能なメタ変数の文字列表現を正規表現で与える．メタ変数は，証明の結合あるいは定理や派生規

```
%ROOT Formula
%META_VARIABLES
  Identifier = "[A-Z][A-Z0-9]*" ;
  Variable = "[x-z][0-9]*" ;
%PRODUCTIONS
  Term ::= Variable ;
  Term ::= Term "/" Term ;
  List_Of_Term ::= Term ;
  List_Of_Term ::= Term "," List_Of_Term ;
  AtomicFormula ::= "⊥" ;
  AtomicFormula ::= Identifier ;
  AtomicFormula ::=
          Identifier "(" List_Of_Term ")" ;
  Formula ::= AtomicFormula ;
  Formula ::= "(" Formula ")" ;
  Formula ::= "¬" Formula ;
  Formula ::= Formula "∧" Formula ;
  Formula ::= Formula "∨" Formula ;
  Formula ::= Formula "⊃" Formula ;
  Formula ::= "∀" @Variable "." [ Formula ] ;
  Formula ::= "∃" @Variable "." [ Formula ] ;
```

図 2 述語論理の構文定義
Fig. 2 Syntax definition of a simple First-Order Predicate Logic.

則を使用する際に必要な構文要素である．*PRODUCTIONS* では，BNF 記法に類似した記法で生成規則を定義する．変数束縛と代入という概念は論理には欠くことができないが，これらも生成規則の中で定義することができる．束縛変数は非終端記号の先頭に "**@**" を付け，その有効範囲は "[" と "]" で括って指定する．図 2 に示す例において，生成規則

Formula ::= "∀" @Variable "." [Formula];

は，記号 "∀"，**Variable** を表す文字列，"." そして **Formula** を表す文字列を続けた文字列は再び **Formula** となることを宣言している．このとき **Variable** を表す文字列が束縛変数であり，**Formula** がその有効範囲である．本表記法により，EUODHILOS を含め，従来の汎用定理証明機では扱いにくい $(\forall x \in A)B(x)$ のような論理式の束縛変数と有効範囲も

```
Formula ::= "(" "∀" @Variable "∈"
        Formula ")" [ Formula ];
```

と定義することで，容易に記述することができる．

一方，代入は通常用いられるように "**/**" によって表現される．この表現は束縛変数の名前換えを行わない，自由変数の部分置き換えを表す．たとえば，"**A(t/x)**" は "**A**" 中のいくつかの自由な "**x**" を "**t**" で置き換えた論理式を表す．この代入表現は公理や推論規則の定義の中でメタ変数とともに使用され，代入可能性を規定する付帯条件 **FREE-FOR**（3.4 節）をあわせて用いることにより，証明の中でそれらを使用する際に，適切な代入が自動的に行われる（4.4 節，導出処理の第 6 ステップ参照）．

また，

```
Formula ::= "(" Formula ")" ;
```

のような形式の生成規則は括弧規則と認識され（"(" と ")" は通常括弧の意味で用いられる他の記号の組でもよい），文字列から内部表現を生成する際には冗長な括弧規則は除去され，論理式の内部構造を文字列に変換する際には論理結合子の優先順位に応じて適宜自動的に補われる．なお，構文定義の中で，1 つの構文要素を定義する構文規則が先に現れるものほど優先順位が高い．

3.2 言語系定義に対する処理

EUODHILOS-II は，構文定義をセーブする際に構文解析用のデータを構文定義から生成し，特定の論理系に依存しない汎用のパーザがそれを参照しながら構文解析を行う方式を採用している．EUODHILOS で採用されたパーザ生成方式と比べ，構文の定義や修正の後，ただちに使用できるため，構文解析のテストを繰り返しながら構文定義を完成させていくのに適している．なお，構文解析には J. Earley の文脈自由文法構文解析アルゴリズム[3]を用いている．

言語定義において，構文定義と他の論理系データとの整合性の保持は重要な問題である．構文定義の修正によって，いったん意図したとおりにパーズされた文字列が意図と異なってパーズされる恐れがあるからである．EUODHILOS-II においては，構文定義の修正後，すべての導出系と証明データをチェックし，誤った構文となった文字列は構文解析し直すことにより，自動的に整合性が保たれる．もし，不整合が発見されたならば，即座に構文定義の修正を行うことができる．

3.3 式の内部表現

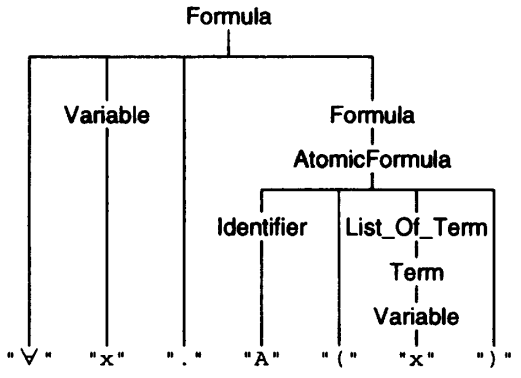
式は図 3 に示されるように構文木を S 式に直接コーディングした内部形式で表される．リストの第一要素は与えられた文字列に最初に適用された生成規則を表す．それ以降の要素は各子ノードの内部表現である．なお，括弧規則に関しては，ユーザが明示的に指定した括弧以外の冗長な括弧は取り除かれる．また，3.1 節で述べた "**@**" や "[" と "]" による束縛変数とその有効範囲に関する情報は，推論規則の適用時や付帯条件の検証時などの際に参照され，式の内部表現には反映されない．束縛変数と有効範囲に関する処理は 3.5 節で述べる．

図 3 の場合，リストの最初の要素は

Formula ::= "∀" @Variable "." [Formula];

という生成規則の S 式による表現である．その後に "∀"，**Variable** に相当する "**x**"，"."，そして **Formula** に相当する "**A(x)**" の内部表現が続く．

この内部構造の採用により，各ノードを生成した生成規則がただちに得られ，その結果，代入処理を行う




```
((Formula "∀" Variable "." Formula) ; 生成規則
  (terminal . "∀")          ; "∀"の内部表現
  (Variable . "x")          ; "x"の内部表現
  (terminal . ".")          ; "."の内部表現
  ((Formula AtomicFormula) ; 以下"A(x)"の内部表現
    ((AtomicFormula
            Identifier "(" List_Of_Term ")")
      (Identifier . "A")
      (terminal . "(")
      ((List_Of_Term Term)
        ((Term Variable)
          (Variable . "x")))
      (terminal . ")"))))
```

図3 "$\forall x.A(x)$" の構文解析木とその内部表現
Fig. 3 Derivation tree of "$\forall x.A(x)$" and its internal structure.

際の代入可能性のチェックや構文木の置換・結合が容易になり，また代入後の構文解析も不要となった．

3.4 導出系定義法

EUODHILOS-II における導出系は，公理・推論規則・書き換え規則によって与えられる．公理は公理図式で与えるが，EUODHILOS と違い，必要に応じて付帯条件を与えることが可能である．推論規則は，結論，前提，仮定を自然演繹法の図式で与え，公理と同様に付帯条件を必要に応じて与える（図4，推論規則エディタ）．形式的には自然演繹法であるが，言語定義においてシークエントを証明の対象にすることで，Gentzen のシークエント計算も定義することができる．書き換え規則は，書き換え前後の式を与えることにより定義される．なお，公理と推論規則に付与することができる付帯条件としては，次の5種類が用意されている．

(1) **FREE-FOR**（代入可能条件）

これは代入に関する条件を規定する．EUODHILOS-II では，"/" が置き換えを意味するため，代入表現を使用する際には，たいていこの付帯条件が必要である．たとえば，

$$\forall x.A(x) \supset A(t/x)$$

という Hilbert スタイルの述語論理の公理の付帯条件は，項 "**t**" を論理式 "**A**" 中の変数 "**x**" に代入する際に，"**t**" の自由変数が代入によって束縛されないことであるが，これは

```
(FREE-FOR ("t" . Term) ("x" . Variable)
          ("A" . Formula))
```

と与えられる．

(2) **NOT-FREE**（変数出現条件）

これは変数がある論理式中に自由な出現を持たないことを規定する付帯条件である．たとえば，

$$\forall x.(A \supset B(x)) \supset (A \supset \forall x.B(x))$$

という述語論理の公理は，変数 "**x**" が論理式 "**A**" 中に自由に出現しないことが必要であるが，それは

```
(NOT-FREE ("x" . Variable)
          ("A" . Formula))
```

と書かれる．

(3) **NOT-FREE-IN-ASSM**（固有変数条件）

この付帯条件は，推論規則に対してのみ有効であり，自然演繹法における固有変数に関する条件を表す．たとえば，推論規則

$$\frac{A(y/x)}{\forall x.A(x)}\ \forall I$$

は "**A(y/x)**" を導くすべての仮定に "**y**" が自由に出現しないという付帯条件が必要であるが，それは

```
(NOT-FREE-IN-ASSM ("y" . Variable)
                  ("A(y/x)" . Formula))
```

と表現される．なお，Gentzen のシークエント計算における固有変数条件は，前項の変数出現条件を使用して表現することができる．

(4) **FULL-SUBST**（全代入条件）

この付帯条件は，公理や推論規則に現れる代入表現が，全代入であることを規定する．たとえば，前項の推論規則は，

```
(FULL-SUBST ("A(y/x)" . Formula))
```

という付帯条件も必要である．

(5) **SYNTAX-CAT**（構文制約）

この付帯条件はメタ変数が表現している構文要素を限定するものである．たとえば，シークエント計算の公理

$$A \vdash A$$

は，A を原子論理式に限定する定式化があるが，その付帯条件は

```
(SYNTAX-CAT ("A" . AtomicFormula))
```

と表される．ただし，**AtomicFormula** は原子論理式

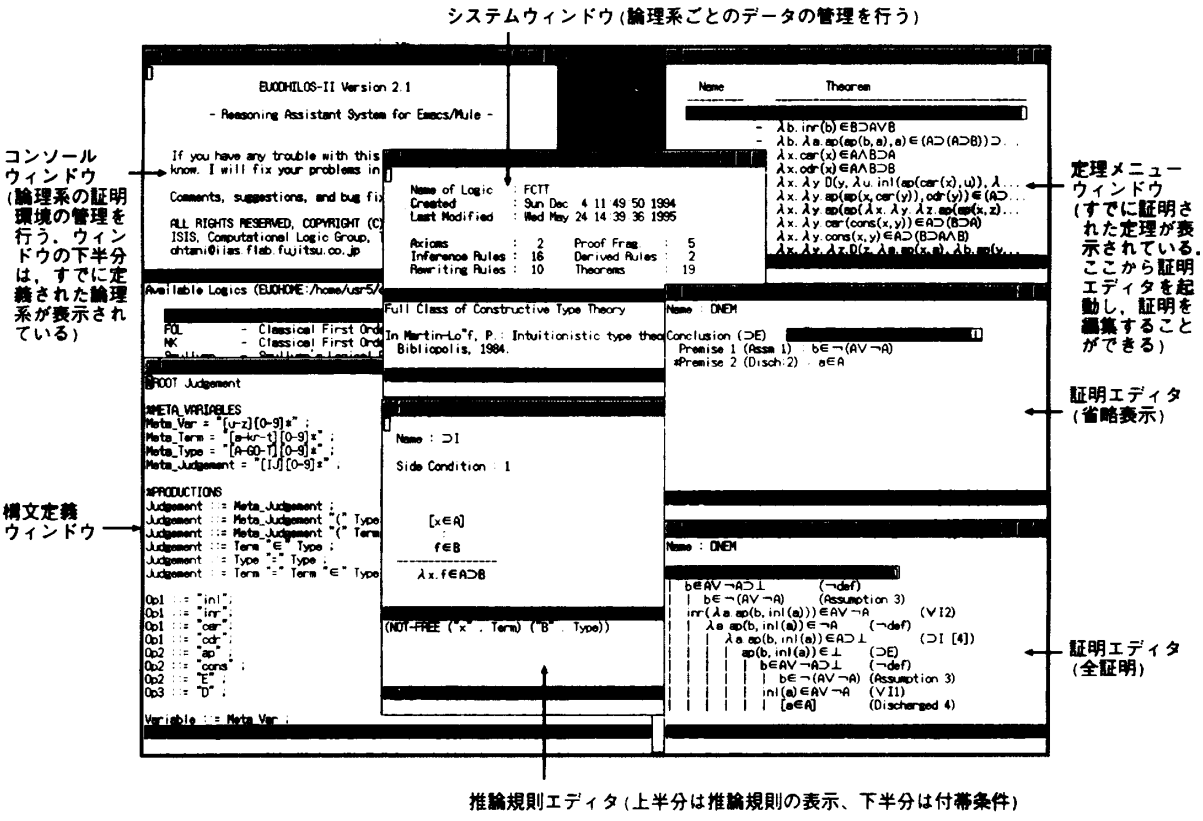


図4 EUODHILOS-II の実行画面
Fig. 4 Sample screen layout of EUODHILOS-II.

のクラスを構文的に表現している非終端記号とする．

この5種類の付帯条件の組合せにより，自然演繹法，Hilbert スタイル，あるいはシークエントスタイルで表現される一階述語論理やタイプ理論などの典型的な論理系における付帯条件を記述できる．

3.5 付帯条件に関する処理

前節で述べた付帯条件のうち，最後の構文制約は式の内部表現から即座に調べることができる．それ以外の付帯条件に関しては，束縛変数とその有効範囲という概念が本質的である．束縛変数と有効範囲が構文定義の中で定義できることはすでに述べたが，ユーザが定義した構文定義からパーザの構文解析用のデータを生成する際に，束縛変数を含む生成規則に対しては，式の内部表現から束縛変数と有効範囲の処理を行うためのデータも同時に生成する．

たとえば，

```
Formula ::= "∀" @Variable "." [ Formula ];
```

という生成規則からは

```
((Formula "∀" Variable "." Formula) 1 3)
```

というリストデータが形成される．最初の要素は，3.3 節で述べた生成規則の S 式表現であり，束縛変数や有効範囲を指定する記号は除かれる．第2要素は束縛変数，第3要素以降は有効範囲の位置を表す．位置は生成規則の子ノードの最初を0番としたときの順番である．この例では，"∀" が0番となり，**Variable** に相当する文字列が束縛変数，**Formula** に相当する部分がその有効範囲と見なされる．変数の出現に関する付帯条件の検証は，式の内部表現の上位のノードから再帰的に行われる．その際に束縛変数を含む生成規則が現れた時点で上記のデータを参照し，有効範囲に相当する部分を走査する場合には，対応する変数を束縛されたものと見なし，検証が続けられる．

3.6 導出系定義の内部表現

3.4 節で述べたように，EUODHILOS-II の導出系定義は，公理・推論規則・書き換え規則の3種類の定義からなるが，本論文では推論規則について説明する．公理と書き換え規則も同様である．自然演繹法における推論規則

$$\frac{\begin{array}{ccc}[\text{仮定式 } 1,1] & & [\text{仮定式 } m,1] \\ \ddots & & \ddots \\ [\text{仮定式 } 1,n(1)] & & [\text{仮定式 } m,n(m)] \\ \vdots & & \vdots \\ \text{前提式 } 1 & \cdots & \text{前提式 } m\end{array}}{\text{結論式}}\ \text{名前}$$

の内部形式は，

(名前 .

```
(付帯条件 .
  (結論式 .
    ( (前提式 1  仮定式 1,1 ... 仮定式 1,n(1))
                    ...
      (前提式 m  仮定式 m,1 ... 仮定式 m,n(m)) ))))
```

となる．“結論式”，“前提式”，“仮定式”は，それぞれ推論規則の結論，前提，その仮定の内部表現であり，“付帯条件”は前節で述べた付帯条件のリストを表す．ただし，各付帯条件の引数に現れる式も，3.3 節で述べた内部表現で置き換えられている．

3.2 節で述べた言語系定義と他の論理系データとの整合性と同様，導出系定義データと証明データとの整合性も，EUODHILOS-II のような汎用システムにおいては非常に重要である．それは，公理や推論規則の変更により，以前に作成した証明，ひいては定理や派生規則が成立しなくなる恐れがあるからである．EUODHILOS-II は，これらの整合性を管理する証明メンテナンス機構として，公理や導出規則の名前の変更に対する証明データの自動修正をサポートしている．

4. 証明構築法の実装

決定可能性や効率の良い定理証明アルゴリズムの存在しない論理系までをも扱おうとする EUODHILOS-II においては，証明方針の決定は人間に任されるため，どのような方法で人間を支援するかはきわめて重要である．EUODHILOS-II の証明構築支援法は，EUODHILOS と同様に，人間の論証スタイルに関する以下の特徴を考慮して設計された．

(1) 導出方向の柔軟性

人間の論証の大きな特徴のひとつは，状況に応じて様々な導出法を自在に組み合わせて行うことである．仮定から演繹的に結論を推論する場合もあれば（前向き推論），結論を導き出すために証明すべき前提を推論する場合もある（後向き推論）．また，それらを取り混ぜて，証明の核となる部分から前提と結論の両方に推論を行う場合や，与えられた仮定と結論の間の推論を補間するという証明の方法をとる場合がある．

(2) 補題・派生規則の利用

大きな問題を 1 ステップずつ証明するのでなく，いくつかのステップをまとめて補題あるいは派生規則という形にし，それらを公理や推論規則のように使い証明を構築する．この作業には 2 つの意味がある．1 つは，いくつかの証明ステップをまとめることで，証明の省力化を図ることである．もう 1 つは，証明の意味のまとまりをつけ，証明を見やすく，そして理解しやすくすることである．

(3) 既成の証明の流用・結合による証明

完全に証明の見通しが立てられる場合は稀であり，たいていは試行錯誤的に証明は行われる．ある方針に基づき進められた証明が失敗に終わる場合もある．そのような場合においても，失敗した証明の一部分が新たな証明構築のための部品として役立てられることがある．また，見通しの立てやすい小さな証明をいくつか構成し，それらを組み合わせて大きな証明を構築することもある．

4.1 EUODHILOS-II の証明構築方法

上記の人間の論証スタイルを支援する環境を実現するために，我々は EUODHILOS-II の証明エディタを，自然演繹法に基づき，以下の基本的機能を持つように設計・実装した．

入力式の柔軟な解釈 ユーザは任意の論理式を“仮定”として証明エディタに入力できる．入力された論理式が実際に仮定として用いられるのか，証明のゴールとして用いられるのかは，ユーザの証明方針に委ねられる．前者の場合は文字どおり仮定と解釈され，前向き推論の前提として利用されることを意味し，後者の場合は後向き推論の結論として用いられることを意味する．また，入力された論理式が公理あるいは定理であるかは EUODHILOS-II によってチェックされ，もしいずれかであれば証明済みの確定された結果として扱われ，前向き推論の際の前提として用いられることが期待される．

メタ変数の利用 必要に応じ随時具体化できるメタ変数を使用した証明を行うことができる．メタ変数を使用することにより，証明の完了した論理式は定理として保存しておき，具体化し他の証明の中で使用することができる．完成していない証明は派生規則として保存し，推論規則と同様に証明の導出に用いることができる．また，メタ変数を含む証明断片の結合による証明も可能である．結合される個所の論理式どうしがパタンマッチされ，2 つの証明の結合を可能とする代入が自動的に求められ，その代入によって証明が具体化され，結合が行われる．こうして，証明断片をモザイクのように組み合わせて規模の大きな証明を作っていくことができる．

対話的証明の支援 EUODHILOS-II における証明構築は基本的に対話的に行われる．証明の各ステップで，ユーザは導出規則の適用対象となるいくつかの証明断片と規則を適用する個所を指定し，さらに推論の方向と適用する導出規則をシステムに知らせる．適用結果が複数得られる場合は，推論可能な証明結果の候

補をすべて生成し，ユーザの選択に任せる．この方式の採用により，導出規則の適用時の煩わしいコマンド入力の負担が大幅に軽減され，入力の間違いも減少する（5.4 節）．

4.2 証明の表示法

証明をどのように表示するかは，対話的な証明機にとって非常に重要である．証明の構造が容易に把握できる表示法により証明の見通しが良くなり，効率的な証明が可能となるからである．この目的のため，EUODHILOS は証明を木構造で表示し，その形式のままの編集が可能である．しかし，証明の規模が大きいと，証明の全体像を把握するのが困難となる．この問題点を解決するために，EUODHILOS-II では，証明木の全体表示と省略表示の 2 種類の表示形式を状況に応じて使い分けることができるようにした（図 4，証明エディタ（省略表示と全証明）参照）．

たとえば，導出規則を適用する場合には，証明のどの部分に対して規則が適用されるのかが最も重要であり，証明の全体構造は必須ではない．前向き推論の場合は証明の結論式とせいぜいそれが依存している除去されていない仮定，後向き推論の場合は除去されていない仮定のみが分かれば，導出を行うのに支障はない．たとえば，証明

$$\dfrac{\dfrac{[A] \quad B}{A \wedge B}}{A \supset A \wedge B}$$

に前向き推論で証明を行う場合，結論の $A \supset A \wedge B$ と，これが依存している仮定 B が重要である．後向き推論の場合には B のみである．しかし，仮定の依存関係のみを表示すると，どの導出規則によって結論が導かれたか不明であるため，ある論理記号に対する除去規則と導入規則を続けて適用し，不用意に冗長な証明をする可能性がある．このような状況の発生を防ぐために，EUODHILOS-II においては，結論と結論が依存している仮定，および結論を導くために用いられた規則を示す証明表示形式を標準としている．

一方，証明の一部を分離し，他の証明で利用，あるいは補題として使用する場合には，対象となる証明の一部分が認識できるように証明木全体を表示する形式でなければならない．このような場合には，証明の表示形式を全体表示形式に切り替えて，操作を行うことができる．

4.3 証明の内部表現

EUODHILOS-II で扱われる証明の内部表現は，次の 3 種類である．

公理・定理：公理および定理は，式の内部表現にそれぞれの識別子と名前が付与されたものである．

((axiom . 名前) . 公理)

((theorem . 名前) . 定理)

定理は完結した証明の結論としてすでに得られ，定理として理論データベースに登録された論理式である．

仮定：仮定には通常の仮定と除去された仮定があり，前者には **assumption**，後者には **discharged** という識別子とラベルが付加されている．

((assumption . ラベル) . 仮定式)

((discharged . ラベル) . 仮定式)

“ラベル” は仮定を区別する識別子（自然数）であり，字面上同じでも異なるラベルを持つ仮定は区別して扱われる．一方，同じラベルを持つ仮定は同一視され，除去される際にも同時に除去される．なお，除去された仮定は仮定を除去する推論によって生成されるため，次項で述べる導出された証明の部分表現としてのみ現れる．

導出された証明：いくつかの証明に導出規則を適用して得られた証明を表す．

(結論式 .

((名前 付帯条件 除去された仮定のラベル) .

(証明 1 ... 証明 n)))

ここで，“名前” は適用された導出規則の名前を，“付帯条件” はその導出規則に付与されている付帯条件を規則の適用時の代入で具体化したリストを指す．付帯条件の検証は，証明途中で随時行うことが可能であるが，その際に各導出ステップの付帯条件の部分が参照され，チェックされる．証明を具体化するときには，この部分も同時に具体化されるため，そのままの形で付帯条件のチェックに用いることができる．“除去された仮定のラベル” は，その導出規則を適用することによって除去された仮定に付与されているラベルのリストである．これは証明の分離を行う際に参照され，該当する仮定に付与されている **discharged** という識別子は **assumption** に戻される．“証明” は導出規則の前提となっている証明の内部表現である．

EUODHILOS-II における証明の内部表現は，証明の全体構造を表現するデータと，証明状態と呼ばれる表現形式の双方からなる．証明状態は，前節で述べた証明の省略表現に対応するもので，各証明ステップで省略された証明を表示し，ユーザからの編集コマンドを受け付ける際に参照されるデータである．そして，このデータは導出が行われるたびに証明の内部表現から生成される．形式は推論規則の内部表現と類似の

(結論式 .
(規則名 .
((前提式 1 仮定式 1,1 ... 仮定式 1,n(1))
...
(前提式 m 仮定式 m,1 ... 仮定式 m,n(m)))))

である．ここで，“結論式” は証明の結論，“規則名” はその結論を導出している規則の名前，“前提式” はその規則の前提となっている式である．“仮定式” は各々の前提式が依存している仮定であり，除去された仮定も含む．式はすべて 3.3 節で述べた内部表現で表される．

4.4 導出処理

EUODHILOS-II における導出方法は，推論の方向（前向きまたは後向き）と適用する規則（推論規則，派生規則，書き換え規則のいずれか）により異なる．本節では，推論規則および派生規則を前向きに適用するアルゴリズムを例として，その概略を示す．

第 1 ステップ： ユーザは導出規則の適用対象となる証明断片と適用個所，および適用する導出規則を指定する．適用個所は，前提式と除去される仮定のリストを要素とするリスト（**証明断片の仮定リスト**）として内部的に表現される．たとえば，2 つの証明断片

$$\dfrac{\overset{V\ \ W}{\Sigma_1}}{U} \qquad \dfrac{\overset{Y}{\Sigma_2}}{X}$$

を選択し，前者では V と W が，後者では Y が除去する仮定として指定された場合，証明断片の仮定リストは，

$((U\ V\ W)\ (X\ Y))$

となる．

第 2 ステップ： 与えられた導出規則の定義データから，前提とその除去される仮定からなるリストを要素とするリスト（**導出規則の仮定リスト**）を作る．たとえば，

$$\dfrac{\begin{array}{ccc}[B] & [C] & [E] \\ \vdots & & \vdots \\ A & & D\end{array}}{F}$$

という導出規則の仮定リストは，

$((A\ B\ C)\ (D\ E))$

となる．それぞれの要素は前項で述べた証明断片の仮定リストに対応する．

第 3 ステップ： 導出規則の仮定リストの前提と仮定の関係を崩さないすべての順列からなるリスト（**仮定順列リスト**）を作る．前項の例では，

$$((A\ B\ C)\ (D\ E)) \Rightarrow \begin{array}{l}(\ \ ((A\ B\ C)\ (D\ E)) \\ \quad\ ((A\ C\ B)\ (D\ E)) \\ \quad\ ((D\ E)\ (A\ B\ C)) \\ \quad\ ((D\ E)\ (A\ C\ B))\ \ )\end{array}$$

となる．これは，前項の導出規則が

$$\dfrac{\begin{array}{ccc}[C] & [B] & [E] \\ \vdots & \vdots & \vdots \\ A & & D\end{array}}{F} \quad \text{や} \quad \dfrac{\begin{array}{ccc}[E] & [B] & [C] \\ \vdots & \vdots & \vdots \\ D & & A\end{array}}{F}$$

のように前提式や仮定式の並ぶ順番を適宜入れ替えて適用されることを考慮し，すべての配列を計算しているのである．

第 4 ステップ： 導出規則の仮定順列リストの要素の中で，指定された証明断片の仮定リストと各要素の長さが同じものを残す．すなわち，推論に必要な前提式と除去される仮定式の個数と，実際に指定された仮定と除去される仮定式の個数が一致するものだけが残される．今の例では，証明断片の仮定リストが $((U\ V\ W)\ (X\ Y))$ なので，それぞれの要素の長さが，3 および 2 である要素のリスト

$$\begin{array}{l}(\ \ ((A\ B\ C)\ (D\ E)) \\ \quad\ ((A\ C\ B)\ (D\ E))\ \ )\end{array}$$

が結果として得られる．もし，この結果が空リストならば，導出規則の適用が不可能として，処理を打ち切る．

第 5 ステップ： 前ステップで得られたリストの各要素と，指定された証明断片の仮定リストのそれぞれの論理式とでパタンマッチングを行い，マッチング可能なものに対し，導出規則の仮定リスト中に出現するメタ変数の具体化を要素とするリスト（**具体化リスト**）を作る．具体化リストは，

$\{p_1 \leftarrow e_1, \ldots, p_n \leftarrow e_n\}$

と表され，これは論理系定義で使用されているメタ変数を表すパタン p_i を具体化式 e_i でそれぞれ具体化することを意味する．このとき，すべてのパタンマッチングに失敗し，結果が空リストとなったならば，第 4 ステップと同様，導出処理を打ち切る．

第 6 ステップ： 導出規則中のメタ変数に代入表現が含まれる場合，第 5 ステップで求められた具体化の整合性のチェックと完備化を行い，不正な具体化があれば削除する．たとえば，

$\{A(x) \leftarrow B(y),\ A(t/x) \leftarrow B(b),\ x \leftarrow y,\ t \leftarrow a\}$

が不整合な具体化の例である．ここで，$A(x)$ は x という表現に注目した A というパタンを表す．よっ

て，$A(x)$ と $A(y)$ はパタンとしては実質的に同じである（"←" の右辺に現れる具体化式として使用する場合は区別する）．これに対して $A(t/x)$ というパタンは $A(x)$ 中のいくつかの x に t を代入したパタンを表す．したがって，$A(t/x)$ の具体化は，原型である $A(x)$ の具体化 $B(y)$ の中のいくつかの y（x の具体化）を a（t の具体化）に置き換えて得られるので，$B(y)$ または $B(a)$ でなければならないが，それは $A(t/x) \leftarrow B(b)$ に反する．また，

$$\{A(t/x) \leftarrow B(a),\ x \leftarrow y,\ t \leftarrow a\}$$

のように，代入表現 $A(t/x)$ に対する具体化が存在するが，原型 $A(x)$ に対する具体化が存在しない場合には，それを計算し補う．これが完備化である．この例の場合，上記の例と同様の理由から，$A(x)$ の具体化は $B(y)$ または $B(a)$ であるので，もとの不完全な具体化は，

$$\{A(x) \leftarrow B(y),\ A(t/x) \leftarrow B(a),\ x \leftarrow y,\ t \leftarrow a\}$$
$$\{A(x) \leftarrow B(a),\ A(t/x) \leftarrow B(a),\ x \leftarrow y,\ t \leftarrow a\}$$

の 2 つの具体化で置き換わる．もし，不正な具体化を削除した結果が空リストならば，第 4 ステップと同様に導出に失敗したことを意味する．

第 7 ステップ：求めた具体化リストに対して，導出規則の結論式に出現するすべてのメタ変数が具体化されるか否かをチェックし，具体化されないものがあればユーザに問い合わせ，それぞれの具体化リストに追加する．

第 8 ステップ：導出規則の付帯条件に，上で求めた具体化を施し，最初に指定した証明断片の中で，付帯条件を満たしているものだけを残す．この場合も，結果が空リストならば導出失敗となる．

第 9 ステップ：具体化リストのそれぞれの具体化を，導出規則の結論式に施す．もし，結論式が一意的に決まらなければ，ユーザに適切な結論式の選択を求める（5.4 節）．たとえば，

$$\frac{\Pi_1}{A} \quad と \quad \frac{\Pi_2}{B}$$

に対して $\wedge$ の導入規則を適用する場合，

$$\frac{\dfrac{\Pi_1}{A} \quad \dfrac{\Pi_2}{B}}{A \wedge B} \quad と \quad \frac{\dfrac{\Pi_2}{B} \quad \dfrac{\Pi_1}{A}}{B \wedge A}$$

のどちらを採用するかをユーザの決定に委ねることになる．

第 10 ステップ：与えられた証明断片に対して仮定の除去を行い，得られた結論式を使用して，新しい証明断片を形成する．

推論規則や派生規則を後向きに適用する場合もほぼ同様であるが，適用対象となる証明断片は 1 つに限られ，また固有変数条件のチェックは行われないために，前向き推論に比べ幾分簡単になる．また，書き換え規則の適用は，後向き推論と同様に適用対象となる証明断片が 1 つであることに加え，適用個所が証明の結論または除去されていない仮定に限定され，推論規則のような仮定の除去を考慮する必要がないため，論理式の書き換えという局所的な操作のみで実現できる．

4.5 証明戦略とその処理

EUODHILOS-II は，EUODHILOS と異なり，証明戦略を利用した証明構築が可能である．EUODHILOS-II の証明戦略は HOL[4] や Isabelle[9] などの定理証明機の証明戦略と同様に，tactic と tactical によって記述される．tactic は証明および導出規則のリストと，必要に応じて適用個所を示すパラメータを引数にとり，その導出規則のリストの中で適用可能な最初の導出規則のみを証明に適用した結果を返す関数であるのが一般的であるが，EUODHILOS-II の tactic は証明のリストと導出規則の集合から証明のリストへの関数で，結果として与えられた証明リストのそれぞれに各導出規則を適用して得られる証明すべてを返す（図 5）．導出規則の集合は名前のパタンを正規表現で与え，それにマッチする名前を持つ導出規則として与えられる．EUODHILOS-II においては，4.1 節で述べたように前向き・後向き推論を自由に適用することのできる証明表現を用いているため，証明戦略による証明構築の際にも，前向き・後向きの両推論を組み合わせて使用できる点が大きな特徴である．また，複数の前提が必要な導出規則を適用する際には，ユーザによって指定された証明断片以外に必要な前提を，公理や定理および他の証明断片から自動的に選択し適用す

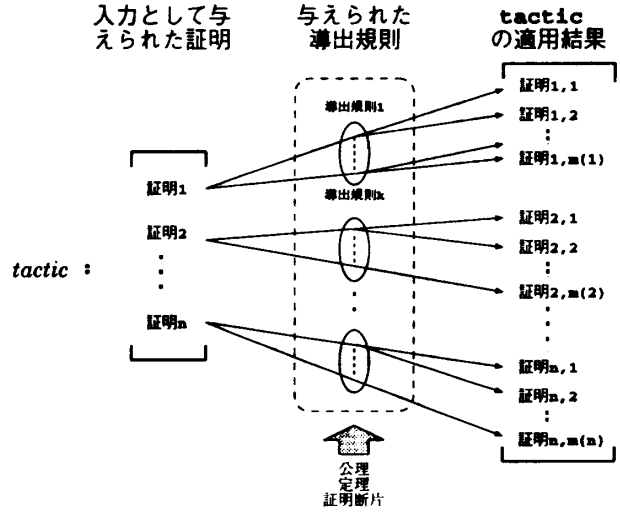


図 5 EUODHILOS-II における tactic
Fig. 5 Tactic of EUODHILOS-II.

る点，さらに証明断片に対して導出規則をどのように適用するかを明示的に与える必要のない点も従来の証明戦略と大きく異なる．一方，tactical は tactic どうしを結合する関数で，各 tactic をどのような手順で適用するかを規定するものであり，HOL や Isabelle などの定理証明機と同様に，**append**，**then**，**orelse**，**repeat** など，リストの連結，証明戦略の逐次実行，条件分岐，繰返しといった基本的な制御コマンドを備えている．

EUODHILOS-II の tactic には規則の適用方法に応じて次のようなものがある．

- **forward_tac**
  証明リストの各要素に対して，引数のパタンにマッチする名前の推論規則を前向きに適用して得られる証明のリストを生成する．もし推論規則が仮定の除去を行うものであれば，除去する仮定の組合せの数だけ証明を生成する．また，複数の前提を必要とする推論規則の場合には，証明対象になっている証明断片は固定し，その他に必要な証明を定理，公理および証明断片の集合から前提として利用可能なものを選び出して推論規則を適用する．
- **backward_tac**
  証明リストの各要素に対して，引数のパタンにマッチする名前の推論規則を後向きに適用して得られる証明のリストを生成する．指定された推論規則が適用可能な個所が複数あれば，その個数分の証明を生成する．
- **rewrite_tac**
  この tactic は証明リストのそれぞれに対して，引数のパタンにマッチする名前の書き換え規則を 1 回適用して得られる証明のリストを生成する．書き換え規則の適用可能な個所が複数あれば，そのそれぞれに対して適用を行う．さらに，その適用可能な論理式中に書き換え可能な部分論理式が複数存在すれば，その個数分の証明を生成する．また，書き換え規則は双方向に適用可能であるため，その両適用方向に対して適用した結果をマージして最終的な結果とする．

証明戦略の適用処理は，対話的な導出規則の適用と基本的には変わらない．異なるのは，適用する導出規則が複数指定され，適用の対象となる証明断片も複数であるので，そのそれぞれの組合せで得られる証明結果を最後にマージする点と，パタンマッチングによって具体化を決定できないメタ変数の具体化を自動的に生成する点である．ここでは，**forward_tac** の処理の概要を述べる．

第 1 ステップ：forward_tac を適用する証明断片の集合と，これらに対して適用する導出規則の集合を **forward_tac** の引数の正規表現から決定する．

第 2 ステップ： 各証明断片と導出規則の組に対して以下の処理を行う．

第 2.1 ステップ： 導出規則の仮定リストの長さを調べ，公理・定理・証明断片から，その数と同じ個数で第 1 ステップで与えられた証明断片を含む証明の組からなるリスト（**証明組リスト**）を生成する．たとえば，導出規則が 2 つの前提を必要とするならば，証明組リストの各要素は 2 つの要素からなり，1 つは指定された証明断片であり，他方は公理・定理・証明断片のいずれかである．

第 2.2 ステップ： 証明組リストの各要素から除去されていない仮定を抽出し，導出規則を適用するのに必要な個数の仮定を備えているものだけを残し，その各々の証明断片に対して規則の適用個所を表す証明断片の仮定リストを作る．

第 2.3 ステップ： 前ステップのデータを基にして，4.1 節で述べた前向き推論のアルゴリズムを実行する．ただし，対話的実行の場合と異なり，パタンマッチングで具体化が決定できないメタ変数に対して，どのような具体化をするかをユーザに問い合わせず，ユーザがあらかじめ指定したプリフィクスに数字を連結したメタ変数を表す文字列で証明中に出現しないものを生成し，使用する．また，すべての適用結果を結果として返す．

第 3 ステップ： それぞれの結果をマージして得られる証明候補をユーザに提示し，ユーザは最終的な結果を決定する（5.4 節）．

たとえば，“**(repeat (forward_tac "⊃ I"))**” は ⊃ の導入規則を前向きに可能な限り適用するという証明戦略である．これを証明断片

$$\dfrac{\dfrac{A^1 \quad A \supset B^2}{B} \quad \dfrac{A^1 \quad A \supset (B \supset C)^3}{B \supset C}}{C}$$

（1，2，3 は仮定を区別するラベル）に適用すると，1 回の “**(forward_tac "⊃ I")**” の適用で仮定の個数分の証明を得，それとともに 1 個ずつ仮定が除去されるので，**図 6** のように，仮定を除去する順番に応じて，合計 6（= 3!）個の証明が適用結果の候補として得られる．

このように導出可能な証明結果を生成する証明戦略は，パラメータを十分に与え厳密な適用を行う従来の証明戦略に対して，証明戦略による証明の推移を完全に予測できない場合にも適用することが可能であり，

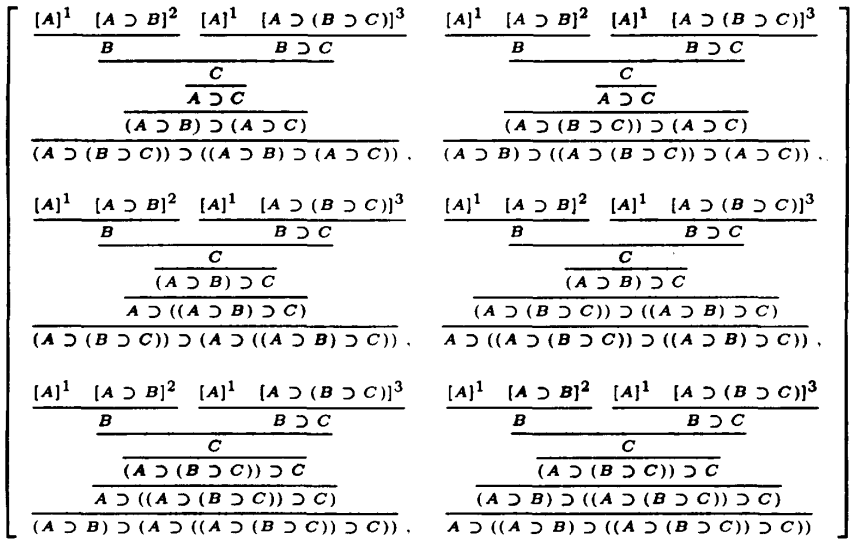


図6 forward_tac の適用で得られる証明
Fig. 6 Resultant proofs by applying forward_tac.

証明戦略を組み立てやすいという利点がある．また，既存の定理や証明などから推論に利用できる証明を検索するため，ユーザが見落としていた証明資源を有効に活用することが可能である．その他，証明を模索，あるいは自分の見落としていた結論を発見するのにも有効である．EUODHILOS-II の証明戦略に関する，これらの特徴は，大規模な証明を自動的に行うというよりも，小規模の自明な証明ステップを補うのに適している．組合せの増大による候補が多数生成される可能性があるという問題点も考えられるが，繰返しなどの単調な証明作業の負担を減らすという証明戦略の目的に照らし合わせると，ユーザへの問合せの発生や証明の中断などの問題点が起こらないため，優れた方式であるといえる．

5. 証明に適したユーザインタフェース

5.1 論理記号の入力

従来の定理証明機の大半は，ASCII 文字しか扱えないために，“→” や “⊃” は “->” に，“∨” は “\/” や “v” などのように，論理記号は適当な ASCII 文字列に置き換える必要があった．これは，ユーザにとって自然で馴染みやすい証明環境を実現する妨げになっていた．また，ωple[2] や PROOF DESIGNER[1] などのように，たとえ論理記号が入力できるとしても，ソフトウェアキーボードによる入力方式のため，入力する文字に応じてキーボードとマウスを使い分けなければならないといった煩わしさがあった．

EUODHILOS-II は ISO Latin 1 文字に論理記号とギリシャ文字の一部を割り当て，論理記号入力のためのフロントエンドを持ち，ユーザはかな漢字変換の要領で容易に論理記号の入力を行うことができる．また，EUODHILOS-II は GNU Emacs の多国語版である GNU Mule でも動作する．GNU Mule には日本語入力の手段として，いくつかのフロントエンドが用意されているため，辞書登録さえすれば，通常の漢字入力と同様の操作法で効率的に論理記号を入力することができる．

5.2 キーボードによる操作

論証を行う際，論理式の入力が頻繁に必要となり，しかもそれはキーボードを通して行われるため，EUODHILOS-II においては，証明の編集作業もキーボードによって行えるように設計されている．コマンドはその機能から容易に連想できるキーに割り付けられている．たとえば，カーソルを次の項目に移動するコマンドは，システム全体で統一的に “n” (next) に割り付け，終了コマンドは “q” (quit) に割り付けられている．EUODHILOS-II の各アプリケーションにおいて実行可能なコマンドの一覧とキーへの割付けは，ヘルプウィンドウによって知ることができる．ヘルプウィンドウは，コマンドの説明のほかに構文定義や付帯条件などの説明も随時参照できる．これらの機能は EUODHILOS-II の操作性の向上に貢献している．

5.3 テンプレートの挿入

EUODHILOS-II は，構文的に不正な論理系定義や証明構築を行うことを防ぐために，基本的には構造エディタとして設計されている．しかし，言語系定義，導出系定義の付帯条件の編集や証明戦略の編集には，編集対象や編集方法に関する自由度があるために，ある特定の構造をユーザに強いるのは適切ではない．たとえば，行頭の段付けや改行の仕方，およびコメント

の付け方などはユーザごとに異なる場合が多く，また編集対象の順番なども画一的に与えられるものではない．そこで，これらを編集するためのエディタには構造エディタを採用していない．しかし，これらのデータにも，当然構文的に正当なものである必要があり，これらのデータを保存する際には，構文的な正当性を検査し，不正な記述はユーザに警告を出し，受け付けないようにしている．

さらに，構造エディタを使用せずに構文的な誤りを少なくするために，編集時に簡単なキーの操作あるいはメニューから構文要素のテンプレートを挿入できるようにした．この機能により，ユーザはテンプレートの引数を書き直すだけですみ，構文的誤りおよびユーザの入力が少なくなり，快適な編集環境が実現された．

5.4 候補の提示・選択

∃ の導入規則のように部分代入が現われる推論規則を適用する際には，実際に代入の行われた個所に応じて得られる証明が複数生じる可能性がある．たとえば，

$$\frac{\Pi}{B(t,t)}$$

という証明に，∃ の導入規則

$$\frac{A(t/x)}{\exists x.A(x)}\ \exists I$$

を前向きに適用する場合，$A(t/x)$ の代入が一般に部分代入を表しているので，実際に代入の行われた変数 x の出現位置に応じて証明結果は次の 4 通りの可能性が考えられる．

$$\frac{\frac{\Pi}{B(t,t)}}{\exists x.B(x,x)}\quad \frac{\frac{\Pi}{B(t,t)}}{\exists x.B(t,x)}\quad \frac{\frac{\Pi}{B(t,t)}}{\exists x.B(x,t)}\quad \frac{\frac{\Pi}{B(t,t)}}{\exists x.B(t,t)}$$

これを一意に定めるために，従来の証明機では，実際に代入が起こった変数の位置をあらかじめ指定して推論規則を適用する方式や，代入が起こる前の論理式の形も同時に与える方式がとられていた．しかし，論理式が複雑になると，指定が繁雑になり，間違いやすくなる．そこで，EUODHILOS-II では，可能な結論式や代入以前の論理式の候補を表示し，ユーザは単にその中から適切なものを選択するだけで証明を進める方式を採用した．この機能は，ユーザの入力とそれにともなう誤りを大幅に減少させ，効率的な証明を行うのに貢献している．

6. 結　　論

本論文では，汎用論証支援システム EUODHILOS-II の仕様設計と実装法を中心に，その有効性について述べた．EUODHILOS-II は，PSI 上に実装された EUODHILOS[6),11),12)] の基本的な設計思想を受け継ぎ，平易な言語系定義法と自然演繹法による論理系定義機能と，柔軟な導出方向，補題・派生規則の利用，既成の証明の流用・結合などの人間の行う証明を容易にするための証明構築支援機能を持つ．EUODHILOS-II は，EUODHILOS の使用経験を基に，さらに以下の改良を施し，再設計されている．

- 束縛変数とそのスコープに関する処理を行うための構文定義法の拡張と，表現力を高めるための正規表現の導入．
- 言語系定義とすでに作成済みの導出系データおよび証明データとの整合性のチェック機能と既存データの自動修正機能．
- 前提となる証明や適用個所などのパラメータを厳密に与えることなしに実行でき，適用結果の候補を生成する証明戦略による半自動証明機能の実装．この証明戦略は，証明を模索する場合，あるいは証明の推移が完全に把握できない場合にも適用できるという特徴を持つ．
- 各証明操作にはそれに適した表明の表示方法があるべきという考察に基づいた証明表現法の採用．
- ユーザの編集操作の自由度と編集対象の正当性を両立させるための編集対象のテンプレートの挿入や，編集操作を容易にするヘルプ機能の実装．
- ワークステーションの標準的なエディタである GNU Emacs 上に実装することによる可搬性・操作性の向上．

汎用の対話型定理証明機は，メタ論理を 1 つ定め，対象とする論理系をその中の理論として定義する方式と，実装言語のデータとして論理系を直接表現する方式とに大別できる．前者に属する代表的な定理証明機には，HOL[4)] や Isabelle[9)] がある．これらは高階論理をメタ論理に据え，その中で論理系を定義できるため，表現力は強力で，表現の正当性の証明も比較的容易であるが，論理系定義の記述は複雑であり，証明構築支援機能は充実しているとはいい難い．これに対して，後者はメタ論理を仮定せず，論理系を直接実装するものであり，ユーザインタフェースにも重点を置くものが多い．この方式を採用しているシステムには，EUODHILOS-II (EUODHILOS) 以外に ωple[2)]，mural[5)]，PROOF DESIGNER[1)] などがある．これらの証明機能の比較を行った結果が**表 1** である．この表より，EUODHILOS-II が他の証明エディタよりも多様な証明の表現や構築方法を提供していることが読み取れる．また，論理記号の入力方法や候補の選択方式は，他の証明エディタには見られず，

表1 EUODHILOS-II と設計思想が類似した他の証明機との比較
Table 1 Comparison of EUODHILOS-II to other proof editors with the similar design principle.

定理証明機	EUODHILOS-II	ωple	mural	PROOF DESIGNER
構文定義	文脈自由文法	文脈自由文法	文脈依存文法	文脈自由文法
証明の表現方法	自然演繹型	シークエント型	自然演繹型	Fitch 型
証明の表示方法	省略形 行形式 木構造	木構造	行形式	行形式
導出手段	公理 推論規則 書き換え規則 定理・派生規則	推論規則 派生規則	公理 推論規則 fold/unfold	推論規則
証明方法	前向き推論 後向き推論 両方向混合推論 結合 証明戦略	後向き推論 証明戦略	前向き推論 後向き推論 両方向混合推論 証明戦略	前向き推論
実装言語	Emacs Lisp	ω-Prolog	Smalltalk-80	Lightspeed Pascal
プラットフォーム	GNU Emacs/X	UNIX/X Window	UNIX	Macintosh

EUODHILOS-II の大きな特徴である.

なお, EUODHILOS-II の最新版は, **ftp://ftp.fujitsu.co.jp/pub/isis/euodhilos2** から入手できる. 我々は, Hilbert スタイルや自然演繹法による一階述語論理, 様相論理, 構成的タイプ理論などの論理系を用いた実験を行い, EUODHILOS-II の有効性を確認してきた. 今後, ユーザからの使用経験を基に EUODHILOS-II を強化するとともに, 次の課題に取り組んで行く予定である.

- 言語系定義の拡張と高速な構文解析アルゴリズムの実装
- 導出系定義と証明との整合性をチェックする証明メンテナンス機能の強化
- 効率の良い探索を行える証明戦略の証明フィルタリング機能と処理系の実装

謝辞 本論文に対して有益なコメントをいただきました査読者の方々に深く感謝いたします.

参 考 文 献

1) Bedau, M. and Moor, J.: *Philosophy and the Computer*, Chapter PROOF DESIGNER: A Programmable Prover's Workbench, pp.218–228, Westview Press (1992).
2) Dawson, M.: A General Logic Environment, PhD Thesis, Dept. of Computing, Imperial College (1991).
3) Earley, J.: An Efficient Context-free Parsing Algorithm, *Comm. ACM*, Vol.3, No.2, pp.94–102 (1970).
4) Gordon, M.: HOL – A Proof Generating System for Higher-order Logic, *VLSI Specification, Verification and Synthesis*, pp.73–128, Kluwer Academic Publishers (1988).
5) Jones, C., Jones, K., Lindsay, P. and Moore, R.: *MURAL: A Formal Development Support System*, Springer (1990).
6) 南 俊朗, 大橋恭子, 沢村 一, 大谷 武: 汎用論証支援システム EUODHILOS 図解マニュアル, Research Report IIAS-RR-92-18J, ISIS, Fujitsu Laboratories Ltd. (1992).
7) Ohtani, T., Sawamura, H. and Minami, T.: EUODHILOS-II on top of GNU Epoch, *Automated Deduction - CADE-12*, LNAI, Vol.814, pp.816–820, Springer (1994).
8) 大谷 武, 沢村 一, 南 俊朗: 論証支援システム EUODHILOS-II: 操作マニュアル, Research Report ISIS-RR-95-4J, ISIS, Fujitsu Laboratories Ltd. (1995).
9) Paulson, L.: *Isabelle: A Generic Theorem Prover*, LNCS, Vol.828, Springer (1994).
10) Prawitz, D.: *Natural Deduction – A Proof-Theoretical Study*, Almqvist & Wiksell (1965).
11) Sawamura, H., Minami, T., Yokota, K. and Ohashi, K.: A Logic Programming Approach to Specifying Logics and Constructing Proofs, *Proc. Seventh International Conference on Logic Programming*, pp.405–424, MIT Press (1990).
12) Sawamura, H., Minami, T. and Ohashi, K.: EUODHILOS: A General Reasoning System for a Variety of Logics, *Logic Programming and Automated Reasoning*, LNAI, Vol.624, pp.501–503, Springer (1992).
13) Stallman, R.: EMACS The Extensible, Customizable Self-documenting Display Editor, *ACM SIGPLAN-SIGOA Symposium on Text Manipulation*, ACM SIGPLAN Notices, Vol.16,

pp.147–156 (1981).

(平成7年12月8日受付)
(平成8年10月1日採録)

大谷　武（正会員）

1964年生．1987年東北大学理学部数学科卒業．1989年同大学院工学研究科情報工学専攻修士課程修了．同年より（株）富士通研究所情報社会科学研究所勤務．論証支援システムの研究に従事した後，現在，協調問題解決の研究に従事．日本ソフトウェア科学会会員．

沢村　一（正会員）

1949年生．1978年北海道大学工学部情報工学専攻博士課程単位修得退学．1980年4月～1996年3月，（株）富士通研究所情報社会科学研究所主管研究員．1989年9月～1990年9月，オーストラリア国立大学客員研究員．1996年4月より，新潟大学工学部情報工学科助教授として勤務．研究分野は計算論理学，ソフトウェア科学，人工知能で，最近はエイジェント指向計算や，議論の論理に興味を持つ．日本ソフトウェア科学会，日本人工知能学会，日本科学哲学会などの会員．

南　俊朗（正会員）

1951年生．1981年九州大学大学院理学研究科博士課程（数学専攻）単位取得退学．1983年より（株）富士通研究所情報社会科学研究所勤務．1992年10月～1993年9月オーストラリア国立大学客員研究員．Agent技術，計算論理学，圏論，人工知能に興味を持つ．情報処理学会，人工知能学会，EATCS会員．