プログラム可視化

所属： ㈱富士通研究所
氏名： 毛 利 友 治

Visualization of Programs

英文氏名： Tomoharu Mohri
英文所属： FUJITSU LABORATORIES LTD.

This paper presents a system to visualize the behavior of ESP programs. ESP is a logic programming language with object-oriented features. A new visualization method embedding tracing probes in a program is proposed. A figure displayed on a workstation can be defined by the user. Animation using figures in functional specification level is helpful to check whether the program reflects its design correctly or not. Hierarchical views and static and dynamic multidimensional views through windows of a display are also useful for the purpose. As an example, a software simulator of a parallel logic programming language KL1 (FGHC) is visualized by our system on a PSI workstation.

1. はじめに

最近のビットマップディスプレイ，マルチウィンドウ機能の発達は，従来不透明であったプログラミングの内部動作を，多元的側面から同時に捉える研究を活発化している．特に，抽象化の手段として重要な役割を果たす図形を有効に使い，プログラムの動作を可視化しようとするビジュアル・プログラミングの研究が盛んになりつつある．不透明なプログラム動作を，より人間の持っている概念に近い形で表示し，理解を助けることは，エンドユーザ開発支援，ソフト資産の蓄積・再利用に大きな効果が期待される．

本論文ではESP で記述されたプログラムの動作を可視化するシステムについて述べる．ESP はオブジェクト指向の特徴を取り入れた論理型プログラミング言語である．[1] ここで述べるシステムはESP のオブジェクト指向，論理型といった2つの特徴を活かしたものとなっている．表示される図形要素とプログラム上のオブジェクトとの間の対応関係はユーザにより定義可能であり，動作表示はプログラムに埋め込まれたトレーサにより抽出された情報をもとに実行される．機能仕様レベルの図形を用いた動作表示は，設計とプログラムの間の一貫性の確認の助けとなる．システムはまた，多重化表示，階層化表示をサポートすることにより，プログラム動作の理解を一層容易なものにしている．

本システムと同様な，多くの試みがある．[2] PegaSys [3]はプログラム設計に対する形式的かつ機械処理可能なドキュメントとして，図形を利用している． PegaSysにおける図形は，アルゴリズムとデータ構造が大規模プログラムの設計にいかにうまくフィットしているかを示す． VIPS [4] はAda のビジュアルデバッガとして開発されている。VIPSではデータの図形表現として，組み込みおよびユーザ定義の2 タイプの図形が利用できる。PROEDIT2 [5]は，PROLOGプログラムのビジュアルデバッガである．PROEDIT2では，Prologの特定の計算モデルBPM に基づいて，box とplane という2 つの図形が用いられる．

我々のESP 可視化システムでは，PegaSys と同様に，設計レベルでの可視化を目指しており，VIPSや PROEDIT2 よりは抽象度の高い表示を行うことが出来る．また，PegaSys や PROEDIT2 と異なり，表示図形は固定的でなく，ユーザ定義が可能である．本論文では，ESP 可視化システムの基本方式，システム概要について述べる．

2. 可視化の基本方式

プログラム可視化システムにおいて表示される画面は，可視化対象であるプログラムの実行時の動作や状態を抽象的に図形表示したものになる．これを可能とするためには，プログラム実行の世界と表示図形の世界との対応関係に関する知識が必要となる。我々のシステムでは，表示図形をユーザ定義可能とすることを目指したため，この対応関係もユーザが定義する．（この対応関係を「可視化指示」と呼ぶことにする．）

プログラムの動作状態から可視化における表示図形の変化を導く可視化方式として，以下の3つの方式が考えられる．

(1) 可視化メソッド埋め込み方式，

(2) インタプリタ方式，

(3) トレーサ埋め込み方式．

(1)は，表示図形の変化が起こると予測した可視化対象のソース・プログラム上の位置に

可視化動作表示のための描画メソッドを直接埋め込み，埋め込んだプログラムの実行によって可視化を行う方式である．この方式の利点として，他方式に比べシステムが比較的単純になることがあげられる．欠点としては，プログラム動作の履歴情報を用意するのが困難であるため，比較的単純な可視化指示にしか対応できないことや，可視化することによる副作用の可能性があることである．

(2)は，可視化対象のソースプログラムをインタプリタ上で実行し，インタプリタから実行情報を抽出し，それを可視化指示によって解釈し可視化する方式である．この方式の利点は，複雑な可視化指示に対しても対応でき，可視化による副作用がないということである．欠点はクロス開発環境や実時間処理のプログラムに不向きなことが挙げられる．

(3)は，上記両方式の中間的方式であり，可視化対象のソース・プログラムに実行情報を抽出するためのトレーサを埋め込み，埋め込んだプログラムの実行によってトレーサから得られる情報を可視化指示によって解釈し，可視化する方式である．この方式の利点は，インタプリタ方式と異なり，クロス開発環境や実時間処理のプログラムにも適用し易いこと，可視化に必要のない情報まで抽出しないで良い点である．また，可視化メソッド埋め込み方式に比べ，プログラムのより細かな動作に対応する可視化指示も扱える．欠点としては，トレーサを埋め込む箇所の判断が可視化メソッド埋め込み方式に比べて複雑となることと，インタプリタ方式でのインタプリタから抽出できる情報と比べるとトレーサから得られる情報は自ずから制限されてしまうことがあげられる．

我々のESP 可視化システムでは，クロス開発環境や実時間処理のプログラムにも適用し易いこと，並列プログラミング言語に対しても適用可能性の有ることを判断理由として，(3)のトレーサ埋め込み方式を採用した。

3. システム概要

3.1 システムの基本構成

図1にシステムの基本構成を示す．プログラム変換部は可視化指示に基づき，ソースプログラム中に，プログラム可視化に必要なプログラム実行情報を抽出するためのトレーサを埋め込む．プログラム動作表示部は変換されたプログラムを実行し，トレーサにより抽出された情報をもとに動作表示を行う．

プログラム動作表示部の詳細な構成を図2に示す．この図では，図1の動作解析系は次の4つの部分からなる．

- プログラムおよび可視化状態の履歴を管理する「履歴管理部」
- トレーサから得た情報に対して処理をおこない，ソース・プログラムに副作用がないようにし，かつ，履歴管理部の履歴と内部可視化指示によりプログラムの事象を解析する「事象解析部」
- この結果と内部形式の可視化指示より内部図形構造の操作を合成し，かつ，履歴を更新する「可視化動作合成部」
- この操作を，内部形式の可視化指示を用いて，対応する図形構造のオブジェクトに対して分配する「分配器」

また，図1のアニメーション系は，次の4つの部分からなる．

- 設計図ライブラリと絵素辞書で定義される図形構造のオブジェクトの「内部図形構造」，複数の設計図やビューに対応して複数の構造を持つ．
- 解析系の図形操作命令から内部図形構造によって解釈されて作られた画面への図形操作を対応する画面に送るための「集配器」
- 表示画面を管理し図形操作をグラフィックスの命令にする「アニメータ」
- 実際のスクリーン上の画面への表示をおこなう「グラフィックス」

さらに，図2全体の制御をユーザとインタフェイスをとりながら行う「ユーザ・インタラクション・ハンドラ」と「システム・コントローラ」が構成として含まれる．

Fig. 1 Overview of the ESP program visualization system

Fig. 2 Overview of the program action visualizer

3.2　可視化指示

可視化指示は，プログラム要素と図形要素との間の写像を定義するものであり，静的な構造およびオブジェクトの対応関係を定義する「写像定義」と，動的な動作の対応関係を定義する「可視化命令」に大きくわけられる．

写像定義は，プログラム上の項およびオブジェクト表現と，絵素オブジェクト表現の対によって，静的な対応関係を定義する．前処理フェーズにおいては，写像定義はトレーサ埋め込み処理の補助情報として使用される．初期化および実行フェーズにおいては，写像定義は内部形式化され後述する分配機構によって，図形構造のオブジェクトへの静的な対応関係として管理される．

可視化命令は，次のようなHoare の公理形表現を用いて定義される．

P｛S｝Q，A

- P：pre-condition　・・・ Sで定まる動作を実行する前のプログラム動作状態，図形状態についての条件．
- S：statement　・・・ 可視化対象のプログラムの実行における述語およびメソッドの呼び出しの動作列．
- Q：post-condition　・・・ Sで定まる動作を実行した直後のプログラム動作状態，図形状態についての条件．
- A：action　・・・ 図形の状態変化をひきおこす命令列．

可視化命令の意味は次のように解釈される。「可視化対象のプログラムの実行において，Sで定まる動作を実行する前にPが成立していた時に，Sで示される可視化のタイミングに実行が達した時点で，Qが成立するならば，Aにより図形状態を変化させる．」

P，Qについては，条件判定をおこなう述語またはメソッド呼び出しであり，Aは，主に図形構造の絵素オブジェクトのメソッド呼び出しである．

Sについては，述語およびメソッドの呼び出しのパターンを記述する．単一の呼び出しに関しては，①その評価の直前，②評価開始直後，③評価終了直後の３つの時点を可視化のタイミングとすることができる．さらに，相互再帰といった複数の呼び出しに関係する実行についても可視化命令を定義できるように，(a)一方の実行後に他方がおこなわれる，(b)一方の評価中に他方が呼ばれる，という指定が可能であるようにした．これにより，ソース・プログラム上で必ずしも連続していないような，catch & throw のように対となる動作に対応しての可視化命令を記述できるようになった．

プログラム変換部おいて，可視化命令はトレーサの埋め込み位置の決定に利用される．埋め込み処理により，どのような可視化指示により，どのようなトレーサをどこに埋め込んだかという情報が得られる．プログラム動作表示部においては，各トレーサに達するごとに，トレーサの埋め込みについての情報と，内部形式で作られる状態履歴とを基に，可視化命令のうち起動可能なものが調べられ，図形の状態変化をひきおこす．

3.3　図形構造

可視化される対象のソース・プログラムと，可視化した画面上の図形との対応がつけば可視化可能である。画面の抽象的構造については次のような性質があるものと仮定する．

- 可視化画面における，ある時間での図形の抽象的・概念的な構成は，実体と実体間の関係で表される構造を持つ。
- 可視化される対象であるプログラムの構造的性質はすべてこの構造に写像される．
- 可視化対象の動作は，この構造の変化，すなわち，画面における図形の変化に写像される．

実際にプログラム可視化システムで扱う実現構造としては，次のように考えた．

- 設計図構造は，絵素オブジェクトを葉とするＤＡＧ構造であり，絵素から設計図を組み上げる階層的構造を表している．
- 構造の一番下を「絵素」と呼び，その上の木のノードは「複合絵素」と呼ぶ．複合絵素のうち，特に，図としての機能がある，すなわち画面（ウィンドウ）への表示の単位と成りうると定義されているものを，「部分（設計）図」と呼ぶ．各設計図構造の根となる部分設計図をとくに「設計図」または「図」と呼ぶ．複数の異なる設計図が存在することを認める．
- 他の設計図を部分設計図として参照するような構造や，部分構造が複数の設計図構造に含まれ，あるいは，共有されることや，同一構造内での共有を認める．

絵素および複合絵素は，「絵素辞書」と呼ばれるライブラリに登録さる．絵素辞書には絵素や複合絵素の形状の正規化された図形的性質，構成だけでなく，自身の形状の変化を伴う正規化されたメソッドも格納されている．絵素辞書には，リスト，キュー，スタックといったユーザが頻繁に使用するソフトウェアの基本データ構造が，システム組み込みの可視化用知識ベースとして用意されている．

設計図および部分図については，「設計図ライブラリ」に登録されている．設計図の構成法は複合絵素のそれと同じであるが，設計図としてユーザが定義するものである．設計図と部分図は，ユーザの定義する階層性により区別される．

プログラム動作表示部において，ユーザ・インタラクションにより設計図が設計図ライブラリより選択されると，定義にしたがって可視化実行部にオブジェクトの木構造をした内部図形構造が作成される．この絵素オブジェクトは木構造の葉ノードにのみなりうる．子を部分として包含する複合絵素および部分図のオブジェクトが任意のノードとなるが，木構造の根ノードは設計図である．複数の木構造が同時に存在可能であり，これらの木構造すべては，設計図プールと呼ばれる単一のオブジェクトにつながれている．

絵素辞書および設計図ライブラリにおいては，単一の要素を異なる構造で共有したり参照することを認めていたが，内部図形構造においては，共有や参照を認めず，厳密な木構造とする．ただし，設計図プールを介した設計図全体の参照についてのみは可能とし，構造を複写することは認める．

3.4　分配・集配方式

トレーサからの情報は，プログラムに副作用を与えないように制御やデータについて絶縁処理がおこなわれた後，解析され，内部図形構造，アニメータへと流れていくが，前節でのべたように，構造が複写されているため，複写された同一の構造への情報も複写してそれぞれに送らなければならない．また，プログラム要素と絵素オブジェクトの間の写像

についても管理しなければならない．内部図形構造は，絵素などが生成，消去されたりしてその構造が変化しうるので，問題は複雑になる．さらに，内部図形構造と画面との対応は，ズーム・インやスクロール，表示／非表示によっても変化する．

このため，内部図形構造への情報の流れ，および，内部図形構造からの情報の流れ，さらに内部図形構造自身の構造について管理をおこなう必要がある．この問題を解決するために，「分配器」と「集配器」とよばれる２つの機構を内部構造の前後にとりつけた．

分配器は，写像情報と図形構造情報を管理することにより，解析系からの情報を対応する絵素または複合絵素に直接送ることができるようになっている．内部図形構造の生成・消滅といった変化は，分配器に委任あるいは報告されておこなわれる．

内部図形構造中で，画面に対する操作は必ず木構造の親ノードに渡され，複合絵素または部分図中で座標変換などの処理がおこなわれる．複合絵素および部分図については，子からの情報を変換してさらに親に送るモードと，子からの情報を無視して自分の図形を親に送るモードがある．このモードの切り替えのように，親に渡す情報を変化させることによって，詳細な内部構造を表示したり，内部構造を省略して表示したりするといった図形上の表現を変化させることが実現できる．

さらに，部分図および設計図は，親または設計図プールに送るだけでなく，集配器に送ることも行う．集配器では，部分図および設計図からの操作に対して操作をおこない，対応する画面に送る．設計図からの図形操作の情報に対して，異なるクリッピングなどの座標変換をおこなうことで，複数の画面に，同一の図の異なるスクロール・エリアを用意することが実現できる．

4. ＰＳＩ上の実験システム

本方式による可視化システムの設計を進めるさいに，実際的な検討をおこなうことを目的として，逐次型推論マシンＰＳＩ上に実験システムを部分試作した．試作システムは以下の機能を持っている．

(1) マルチ・ビュー： プログラム動作を複数の設計図（マルチ・ビュー）に対応したウィンドウ上で同時に可視化する．その際に，ユーザ・インタラクションとして，可視化の中断・再開，各設計図の表示，非表示の切り換え等の機能を用意した．これによりプログラムの動作状況を複数の設計図上で同時に可視化し，かつ，任意の時点で可視化を一時中断させて全体を把握することが可能となり，可視化の効果を向上させることができた．

(2) 基本データ構造： システムが提供する基本データ構造として，アレイ，スタック，キュー，リスト等に対する操作と形状について検討し，そのなかで使用頻度の高いキューとリストを試作した．

(3) 簡略表示： データ構造に大量の要素が格納される場合，データ構造内に格納されている要素を簡略して表示する機能を検討し，試作した．この機能は，設計図中でデータ構造を表示するスペースが限られている場合でも，そのスペース内で大量の要素を表現できるため，非常に有用であることが確認された．

実際的な例題プログラムとして，KL/I分散処理系の基本モデルをシミュレートするKL/I

シミュレータ[6]についておこなった可視化例を図3に示す．この例では，KL/Iシミュレータの全体構成を表す設計図と，その中に出現する複数のプロセッサ（Processing Element，PE）の内部構成を示す設計図を用いて，シミュレータの動作が可視化されている．また，プロセッサ#1の内部構成図'PE#1'では，スケジューラ(Sched)とソルバ(Solve)の間でキューが形成され，そのキューは簡略表示機能を用いて表示されている．

〔謝辞〕本論文は第5世代コンピュータシステムプロジェクトの一部として実施された研究に基づいている．研究の機会を与えていただいたＩＣＯＴの長谷川第一研究室長に感謝致します．

〔参考文献〕

(1) Chikayama, T., "Unique Features of ESP", Proc. of FGCS'84, pp.292-298 (1984)

(2) Grafton, G.B. and Ichikawa, T., "Visual Programming", IEEE Computer, Vol. 18, No. 8, pp.6-9 (1985)

(3) Moriconi, M. and Hare, D.F., "Visualizing Program Design Through PegaSys", IEEE Computer, Vol. 18, No. 8, pp.72-85(1985)

(4) Isoda, s., Shimoura, T. and Ono, Y., "VIPS: A Visual Debugger", IEEE Software, pp.8-19 (May 1987)

(5) Morishita, S. et.al., "Prolog Computation Model BPM and its Debugger PROEDIT2", Lecture Notes in Computer Science 264, pp.147-158, Springer-Verlag, Berlin(1987)

(6) Ohara, S. et. al., "A Prototype Software Simulator for FGHC", Lecture Notes in Computer Science 264, pp.46-57, Springer-Verlag, Berlin (1987)

Fig.3 An example of visualization of KL/I software simulator

汎用論証支援システム　EUODHILOS
―その機能と実現システムについて―

General-Purpose Reasoning Assistant System
EUODHILOS
―Function and Implemented System ―

② 著者名・所属

＊佐藤かおる Kaoru Satoh 会員番号8521945

大橋恭子 Kyoko Ohashi 会員番号8705826

富士通㈱

Fujitsu Ltd.

沢村一 Hajime Sawamura 会員番号7604220

南俊朗 Toshiro Minami 会員番号8312918

富士通㈱国際情報社会科学研究所

International Institute for Advanced Study of Social Information Science, Fujitsu Ltd.

連絡先

〒211 川崎市中原区上小田中1015

㈱富士通研究所 ソフトウェア開発部第2開発課

☎ 044-777-1111 (内線 2-6156)

③ 概要

問題解決を行うには，それに適した論理の下で証明を行えることが望ましい．しかし，これまでに研究，開発されてきた証明（支援）システムには，特定の論理系の下での支援を行うものが多い．そこで，我々は，論理系を固定しない汎用な論証支援システムを目標として，EUODHiLOS (Every Universe Of Discourse Has Its LOgical Structure) を開発した．本システムでは，ユーザの定義した論理系の下での証明構築を支援する．論理系は，言語系の定義として，記号および論理式の構文，導出系定義として，公理と導出規則（推論規則と書き換え規則）を定義することで定められる．証明は，いくつもの証明断片を作り，それらを結合，分離しながら構築していくという方法で行うことができる．本システムは，ユーザの思考に沿って支援が行えるように実験，修正が容易に行えるような柔軟性を持つ対話型システムである．また，通常，論理を扱う上で自然な表現を用い，論証に適したユーザ・インタフェースを提供している．ユーザが理解しやすいように，証明，論理式などを視覚化し，これらに対する操作も表示された図の上で可能である．本論文では，本システムの機能とそれらを実現したシステムについて述べる．

④　本文

1.　はじめに

今日，計算機科学や人工知能の分野において，様々な論理およびその応用の研究が盛んである[1]．論理の下で行う問題解決とは，問題の形式化，証明を経て，定理を導くことである．このような過程を計算機上で支援することは，求める問題解決や理論構築を正確かつ容易に行えるようになるので，有益である．

我々は，計算機上で，人間の記号および論理操作を支援することを目的として，論証支援システムを開発した．本システムは，次の３つの特徴を持つ．

(1)　論理系を固定しない汎用の論証支援システム

我々は，「すべての議論領域は，それぞれの論理構造を持つ」[2]という思想をシステムの設計思想の核に置いた．すなわち，論証は問題に適した論理系の下で行うべきであるという主張をシステムに実現することを目標とした．これまでに研究されてきた証明（支援）システム[3][4]の多くは，特定の論理系の下での証明を行うことを目的としている．しかし，問題毎にそれに適した論理系の下での支援システムを個々に構築するのは，その度ごとに類似の労力を要し，無駄が多い．一方，すべての議論領域を包含するような一つの論理系を定めることは難しく，また可能であったとしても個々の論理系の特性を活かすことができない．そのため，それぞれの論理を規定できるメタな枠組みを

提供し，その上で各論理を定義することが有効である．我々は，ユーザが解決しようとする問題に適した論理系を定義し，その論理系の下での証明構築の支援を受けることができるようなシステム，すなわち論理系を固定しない汎用の論証支援システム[1]を開発した．

(2) 人間の思考に沿った操作を可能にする柔軟性

論証の過程は，適切な論理系を定め，問題を解決する証明を得るまでの試行錯誤の過程である．このような過程を支援するシステムには，ユーザの思考に沿って，実験，修正を容易に行える柔軟性が必要である．本システムでは，定義した論理系の下で求める証明が行えるかどうかを実験し，再度論理系を修正するといった試行錯誤が容易に行えるような支援を行う．本システムは対話型システムで，問題解決の結果のみでなく，その過程をも重視した．

(3) 論証に適したユーザ・インタフェースの提供

支援システムのユーザ・インタフェースは，ユーザの思考を助ける上で重要である．本システムは，論証を行う際に，通常，論理を扱う上で自然な表現と操作を提供し，紙と鉛筆の代わりとなるような論証に適したユーザ・インタフェースを持つことを目標した．論証の過程では，複雑な証明や論理式を扱うので，これらをわかりやすく図で表示し，図的表現に対する操作を簡単に行えるようにした．

図の上で操作を行うには，操作対象に対してラベルを付け，そのラベルを用いて対象指定するよりも，表

示されている対象そのものをマウスで指定できる方が望ましい．本システムでは，マウス主体でコマンド入力を行い，複雑な指示には，アイコン，メニューなどを組み合わせて入力する．

本稿の目的は，上記の要求を満足するための汎用論証支援システムEUODHILOS (Every Universe Of Discourse Has Its LOgical Structure) の機能とそれらを組み込んだ実現システムを提案することである．

以下，第2節で，本システムの概要について，第3節で，汎用性を実現するための論理系定義支援について，第4節で，ユーザの思考に沿って，ユーザ・フレンドリな支援を行うことを目的とする証明支援について，述語論理[1]の例を用いて説明する．そして，第5節で他の論理系を定義し，その下での証明を行った例を挙げる．なお，本システムは，ICOTで開発された逐次推論マシンPSI 上でESP 言語によって実現した．

2. システム概要

図1に，本システムの構成を示す．本システムは，主に，ユーザが論理系を定義するのを支援する論理系定義部と定義された論理系の下での証明を支援する証明支援部から構成されている．論理系定義は，言語系定義と導出系定義から成る．言語系定義がなされると，それをもとにパーザ生成部がパーザを生成する．定義される論理系の下での論理式は，すべてこのパーザに

よって解釈される．パーザで解釈された論理式の構造は，構造単位で操作が可能な論理式エディタで表示および編集を行うことができる．導出系定義は，証明支援部で導出を行うときに参照される．論理系が定義されると，証明支援部の思考シートでは，その論理系の下での証明の支援を行う．定義とその下での証明や定理は，論理系毎に理論データベースに格納される．証明を簡略化する手段として，既に証明された定理を証明の出発点として用いたり，複数ステップの導出を一つの派生規則として登録して使用したりすることができる．

3. 論理系定義支援

汎用性を実現するために，本システムは，論理系を定めるメタな枠組みを提供する．論理系の定義には，言語系と導出系の定義が必要である．言語系定義は，定義する論理系の下で自然な形で論理式を表現できるために必要な定義で，記号の定義（作成）と論理式の構文の定義から成る．導出系の定義は，公理と導出規則（推論規則，書き換え規則）から成る．論理系定義支援部では，これらの定義を行うための各機能を提供する．

3.1 言語系定義

(1) 記号定義とソフトウェア・キーボード

論理の世界では，それぞれの論理系に特有な記号を用いる．例えば，命題論理では論理記号として∧，∨

などを使用するし，述語論理では限量記号として∀，∃などを使用する．このような記号は，通常，キーボードには用意されていないため，各論理毎に必要な記号をフォント・エディタで作成する．作成した文字はキーボードのキーに割り当てることで入力が可能となる．キーの割り当ては，画面上に表示されるソフトウェア・キーボードに表示される．

図2は，述語論理に必要な記号（～，∧，∨，⊃，≡，⊥，∀，∃）を作成し，割り当てたソフトウェア・キーボードである．

(2) 構文定義

① 論理式のための構文定義法

本システムでは，それぞれの論理系に固有な表現を用いるために，論理式の構文を固定しない．そのため，システムで必要な論理式の構文解析系は，ユーザが定義した論理式の構文をもとにパーザを生成することで用意する．論理式を一般的な形式で表現し，すべての論理系で同じ表現を用いる方法も考えられるが，この場合，ユーザは通常親しんでいる自然な表現を用いることができないという欠点がある．

論理式のための構文定義とパーザは，次の2つの要件を満たす必要がある．1つは，論理式の構造も定義でき，解析できること，もう1つは，オペレータなど構文要素の位置が任意であることである．例えば，Prologのオペレータ宣言[1]のように，優先順位と前置，後置，中置（右結合，左結合）の区別で定義する方法

では，オペレータの位置に制約があり，適当ではない．

これらの要件を満足させる定義方法として，本システムでは構文の定義を確定節文法（DCG）[8]を採用した．DCG では，通常，再帰的に定義される論理式の構文や構造を自然に表現することができる．例えば，命題論理の論理式の構文が次のように与えられたとする．[9]

・基本命題 p,q,rは論理式である．

・P が論理式であるとき，～P も論理式である．

・P,Q が論理式であるとき，P—Q も論理式である．

このとき，本システムでは次のように定義を行う．

```
formula --> formula, imply, formula1;
formula --> formula1;
formula1 --> not, formula1;
formula1 --> basic-formula;
basic-formula --> "p" | "q" | "r";
imply -->  "—";
not -->  "～".
```

ここでは，論理式の非終端記号にformula とformula1 を導入し，imply を含む論理式は，左再帰的な構造を持ち，かつnot を含む論理式よりも結合が弱いことを表現している．

本システムでは，次に述べるパーザ生成に必要なため，各節中でのオペレータを構文定義の最後に次のように宣言する．

```
operator    not, imply.
```

図３は，本システムで述語論理の論理式の構文を定

要し，一部を表示した例である．

② メタ記号の定義

本システムでは，論理式にスキーマも扱えるように，任意の論理式や項を代入できるメタ記号を定義することができる．論理式スキーマは，導出規則，公理図式，定理図式などに用いられ，パターン・マッチングや代入などによって，具体化される．

(3) パーザ生成

構文が定義されると，システムはこのデータから論理式用ボトム・アップ・パーザ[10]を生成する．通常のトップ・ダウン・パーザには，左再帰的な定義をすることができないという欠点がある．論理式の構造としては，左再帰的な構造も扱えることが望ましいため，ボトム・アップ・パーザを採用した．

本システムでは，論理式を構造単位で扱うため，解析した結果として，構造データが必要である．そのため，パーザは，構文チェックを行うだけではなく，論理式を構造データに変換する必要がある．通常のDCGの記法では，このような付加的な情報は引数で表現し，ユーザが記述しなければならない．そこで，本システムでは，ユーザが構文定義を行う際の負担を軽減するため，パーザ生成時に，構造データに変換する機能を自動的に付加する．

また，システムは構造データを操作するため，操作後生成される論理式をユーザへ出力するための文字列に変換する必要がある．そのため，パーザを生成する

のと同時に同じ構文定義をもとに構造データから文字列への変換を行うプログラムを自動的に生成している．

言語系の定義を行い，パーザが生成されると，論理式の入出力を行うことができるようになる．

3.2　導出系定義

(1)　公理

公理は，証明の出発点となる論理式で，メタ記号を用いる公理図式を含む．公理図式は，メタ記号に適当な論理式を代入して具体化できる．

(2)　推論規則

推論規則とは，いくつかの前提から一つの結論を導く規則をいう．推論規則は，前提の持っている仮定（適用後に結論の依存する仮定から除かれるもの）や適用条件を付けて定義することもできる（仮定が付いている場合の解釈は，自然演繹法[6]に準ずる）．本システムは，適用条件に関して，2つの問題点を残している．1つは，汎用の論証支援システムであるため，適用条件記述言語が必要であるということ，もう1つは，第4節で述べる証明方法論上，チェックが難しいということである．今後，この点に関する検討が必要である．現在は注釈として記述し，規則適用時に警告を出して，ユーザが確認する．推論規則定義は，次のように横線で導出を表す．

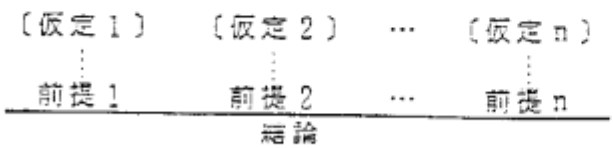

図４は推論規則の定義例である．図中，Ｐ，Ｑで表される記号は，メタ記号である．図４の例は，ＩＭＰＩという名前の推論規則で，仮定Ｐを持つ前提Ｑから結論Ｐ⊃Ｑを導き，仮定Ｐを結論の依存する仮定から除くことを表している．

(3)　書き換え規則

書き換え規則とは，ある表現を別の表現に書き換える規則をいい，論理式全体だけでなく，部分項にも適用可能である．書き換え規則は，次のように，書き換え前と書き換え後の対で表現する．

書き換え前の表現

書き換え後の表現

図５は書き換え規則の定義例である．この例は，～～Ｅという名前の書き換え規則で，～～Ｐが書き換え前，Ｐが書き換え後の表現である．

4.　証明支援

論理系の定義が行われると，システムは，これらの定義に基づいて，証明支援[1]を行う．

証明支援部では，人間の思考に沿って支援を行うための柔軟な証明方法を実現する対話型の証明コンストラクタを提供する．また，複雑な構造を理解しやすくするため，証明や論理式は図的表現で扱うことができる．

4.1　定理証明

証明とは，証明の出発点となる論理式（仮定，公理，

既に証明済みの定理）に推論規則や書き換え規則の適用を繰り返して，定理を導くことをいう．本システムでは，一般的な問題に適用可能な部分やまだ具体化できない部分を保留しておき，後で具体化できるようにメタ記号を用いて定理図式を証明することができる．得られた定理および定理図式は，証明と共に理論データベースに格納し，蓄積できる．蓄積された定理とその証明は，同じ論理系の下で他の証明を行うときの参考となる．

4.2　思考シート

証明支援部では，証明を行う上で柔軟な環境，すなわち，ユーザが好きなところから手をつけ，好きな方法で考え，やり直しが容易にできる環境を提供することを目標とした．本証明支援部では，証明の途中から始めたり，役に立ちそうな中間結果までを証明したりして，証明断片を作成し，それらを結合したり，分離したりすることで証明を構成していくといった証明方法が可能である．また，思考シートでは，一つの証明を構成できそうな証明断片や関連のありそうな証明断片をグループ化できるように，複数のウインドウを複数のメモ用紙の比喩として使用できる．そして，ウインドウ間の証明断片や論理式の移動，複写が容易に行える．我々は，このような証明支援環境を思考シートと呼ぶ．

4.3　思考シートにおけるユーザ・インタフェース

証明は，仮定や公理をリーフとし，導出結果を各

ノードに持つ木構造を持つ．思考シートでは，これを証明図[6]として表示する．図6は，証明図の例である．この証明図では，〔∃xA(x)〕，〔∀x ～A(x)〕，〔A(a)〕を仮定として，∀x ～A(x)⊃～∃xA(x) が証明されている．横線は導出を表し，横線の右端には適用した規則と導出結果の依存している仮定の番号が表示される．また，論理式に対しては，文字列編集の他に，定義された構文に基づく構造を表示し，その構造単位に編集が可能な論理式用構造エディタ（論理式エディタ[12]）を提供している．図7は，述語論理の論理式∀x ～A(x)⊃～∃xA(x) を論理式エディタに表示したものである．

4.4　思考シートの機能

(1)　証明の出発点の設定

証明の出発点としては，仮定，公理（前提），定理がある．仮定は，予め設定しておくのではなく，証明に応じて思考シートに思いついた時点で書き込める．入力された仮定式は，仮定式であることを表すために〔　〕で囲まれ，それぞれに識別番号が付けられる．この番号は，導出結果の依存する仮定の番号を表すときに用いられる．公理は論理系定義の際に与えられており，定理は，思考シートで証明済みのものが，論理系のデータベースに格納されている．これらはメニューで選択し，引用する．

一旦入力された論理式は，編集，変更が可能である．論理式を指定すれば，その論理式を編集することがで

きる．また，論理式単位に置き換え，削除，複写，移動などの操作が可能である．

(2) 前向き導出，後ろ向き導出

試行錯誤的な証明の進め方には，仮定や既に証明された結論などから新しい結論を導く方法，目的とする結論からその結論を導くための中間結果を導く方法などが混在している．この両方法を用いて証明断片を作り，各証明断片の間を埋めることで証明を完成させていく．

導出は，前提，結論，導出規則の3つから成り立っている．このうち2つを指定すると，システムは導出規則を参照してあとの1つを求めることができる．思考シートでは，ユーザが前提と導出規則を指定してシステムが結論を出力する前向き導出と，ユーザが結論と導出規則を指定してシステムが前提を出力する後ろ向き導出の両方向の導出を支援する．また，ユーザが前提または結論を指定して，導出結果を入力すれば，システムが適用可能な導出規則を提示する．

仮定付きの推論規則を適用する場合には，前提の他に仮定を指定する．しかし，仮定の指定は，保留しておき，後で指定することも可能である．後ろ向き導出を行った場合は仮定を指定しないので，保留した場合と同じ扱いになる．仮定の指定を保留した場合は，導出が不完全である印として導出されたステップの横線の右端に＊が表示される．後に仮定を指定して除いた場合には，＊の表示が除かれ，そのステップの導出が

完結する．

(3)　証明断片の結合，分離

証明を進めていくと，複数のシートにいくつかの証明断片ができあがる．証明の目的は，そのような証明断片，すなわち部分証明から証明全体を構成していくことにある．

思考シートでは，ある証明断片で他の証明断片の仮定を導出している場合，このノードで証明を結合し，より大きな証明を構成することができる．逆に，ある証明の途中までを利用して，他の証明を作ったり，展開させたりするために，証明をあるノードから分離することもできる．

結合，分離の際，既に仮定から除かれていた仮定が仮定でなくなってしまった場合は，導出が不完全になるので，＊が表示される．

(4)　論理式の削除

正しい証明を得るのに不都合な論理式は，証明を構成せずに独立している場合と証明のリーフまたはルートにあたる場合に削除することができる．証明図を構成する論理式（証明のリーフまたはルート）を削除した場合には，導出が不完全にならないように，削除された論理式を使用する導出のステップが取り消される．取り消された導出が削除された論理式の他に部分木を持っていた場合には，その部分木が分離される．

4.5　派生規則

本システムでは，証明の簡略化を目的として，複数

の証明ステップを1つの派生規則として登録することができる．登録後は，その派生規則を導出規則と同様に使用することができる．派生規則は，思考シートで証明を行うのと同様に図式表現の論理式を用いて証明を行い，規則名を付けて登録できる．

5. 他の論理系定義および証明例

システムの有用性を確かめるために，現在までに，代表的な論理として，述語論理，様相論理，内包論理，Hoare 論理，dynamic 論理などの定義を行い，その下での証明を行った．図8は，様相命題論理を定義し，プログラムの検証に用いられる論理式を証明した例である．様相命題論理は，命題論理に様相オペレータ◇，□を導入した論理である．定義は，命題論理をもとに若干の追加を行えばよい．図9は，Martin-löfの直観主義タイプ理論を定義し，排中律の二重否定を構成的に証明した例である．Martin-löfの直観主義タイプ理論は，論理式中にλ式を用いており，λ式やタイプに関する導出規則を持つことを特徴とする．

6. おわりに

本論文では，汎用論証支援システムGUODHILOS の機能とそれらを実現したシステムについて述べた．最近，我々の研究と類似した研究がいくつか見られる[13][14]．それらの研究の重点は，主に論理の一般理論を目指した論理の記述法の研究に置かれている．一方，我々の

研究は，論理系の定義機構による汎用性の実現に加えて，様々な証明構成支援機構（証明方法論）をも初めから組み込んだシステムとして特徴づけられる．

本システムの汎用性については，まだ実験中であるが，今までに定義を行った論理については，既に第3節で述べた．現在，1つの論理について数時間程度で定義および証明を行うことができる．以後，本システムを用いた場合に定義できる論理系の範囲を考察し，更に広範囲の論理系を定義できるよう，機能拡張を行いたい．

本システム開発にあたって，柔軟性および論証に適したユーザ・インタフェースということに重点をおいて，設計を行った．その結果，論理系構築および証明構成において，人間の思考に沿った方法論に基づく実験および修正の容易な支援システムを実現した．しかし，このような動的なシステムでは，証明の整合性のチェックを行うタイミング，動的に変化する論理系とその下で蓄積された定理，証明などとの整合性などの問題があり，今後の課題となっている．ユーザ・インタフェースの他の側面に関しても，今後，使用経験を通して更に検討および改良を行う予定である．

なお，本研究は，第5世代コンピュータ開発の一環として，ICOTの委託によって行ったものである．

⑦　謝辞

日頃，御指導いただく，委員諸氏に感謝いたします．

⑥ 参考文献

1) Turner, R. : Logics for Artificial Intelligence, Ellis Horwood Limited (1984).

2) Langer, S. K. : A set of postulates for the logical structure of music, Monist 39, pp. 561-570 (1925).

3) ICOT : ICOT CAP-WG : CAP プロジェクト(1)～(6), 情報処理学会第32回全国大会 (1986).

4) Ketonen, J. & Weening, J. S. : EKL - An Interactive Proof Checker, User's Reference Manual, Dept. of Computer Science, Stanford University (1984).

5) 南, 沢村, 佐藤, 土屋, 菱 : 論証支援システム : 論理モデル構築のための支援ツール, Proceedings of the Logic Programming Conference 88', ICOT (1988).

6) 前原昭二 : 記号論理学入門, 日本評論社 (1967).

7) Clocksin, W. F. & Mellish, C. S. : Prolog プログラミング, マイクロソフトウェア (1983).

8) 溝口文雄他 : Prologとその応用, 総研出版 (1985).

9) 長尾真, 淵一博 : 論理と意味, 岩波書店 (1982).

10) Matsumoto, Y., Tanaka, H., Hirakawa, H., Miyoshi, H. & Yasukawa, H. : BUP : A bottom-up parser enbedded in Prolog, New Generation Computing, Vo. 1, pp. 380-388 (1987).

11) 南, 沢村 : 落書帳からの証明－計算機による論証支援のための証明方法論の一考察－, 情報処理学会第

35回全国大会 (1987).

12) 佐藤, 小野, 小野, 沢村, 南：論証支援システムのための論理式エディタ, 情報処理学会第33回全国大会 (1986).

13) Harper, R., Honsell, F. & Plotkin, G. :A framework for defining logics, Proc. of symposium on logic in Computer Science, pp. 194-204 (1987).

14) Felty, A. & Miller, D. : Specifying theorem provers in a higher-order logic programming language, LNCS 310, pp. 61-80 (1988).

⑥ 図

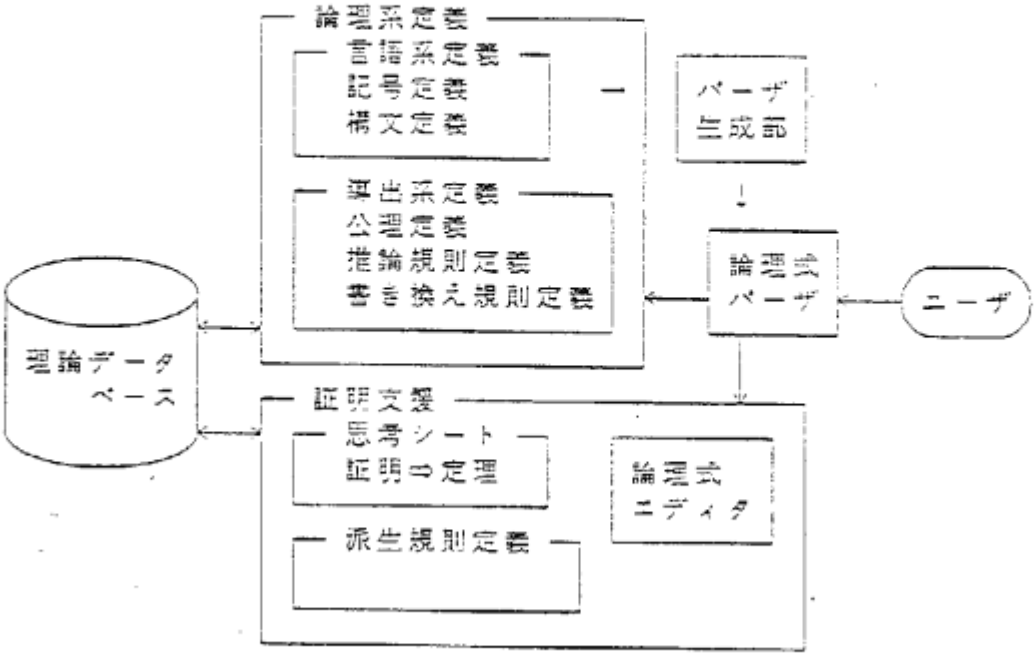

図1　システム構成

Fig.1 System configuration

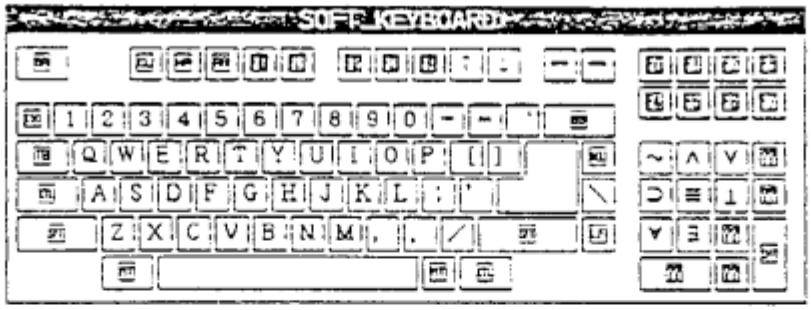

図２　ソフトウェア・キーボード

Fig. 2 Software keyboard

```
SYNTAX = pred
formula --> formula, imply, formula1;
formula --> formula1;

formula1 --> formula1, or, formula2;
formula1 --> formula2;

formula2 --> formula2, and, formula3;
formula2 --> formula3;

formula3 --> "(", formula, ")";
formula3 --> not, formula3;
formula3 --> bind_op, individual_var, for!
mula3;
formula3 --> basic_formula;

bind_op --> "∀";
bind_op --> "∃";
imply --> "⊃";
or --> "∨";
and --> "∧";
not --> "~";

basic_formula --> predicate_symbol, "(", !
term, ")";
basic_formula --> term, equal, term;

predicate_symbol --> "A";
predicate_symbol --> "B";
predicate_symbol --> "C";
```

図３　構文定義

Fig. 3 Syntax definition

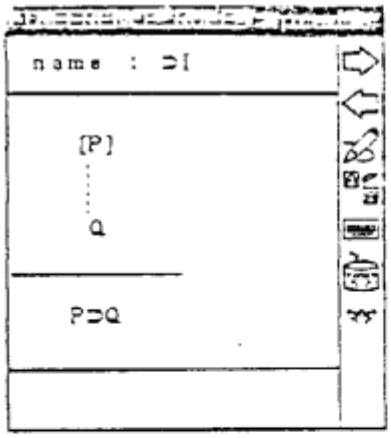

図 4　推論規則定義

Fig. 4 Inference rule definition

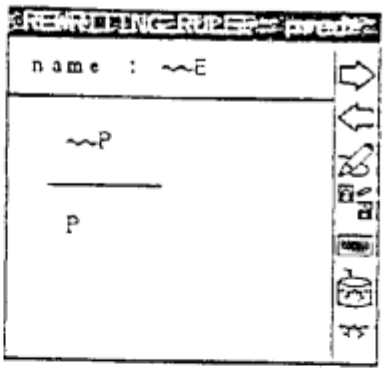

図 5　書き換え規則定義

Fig. 5 Rewriting rule definition

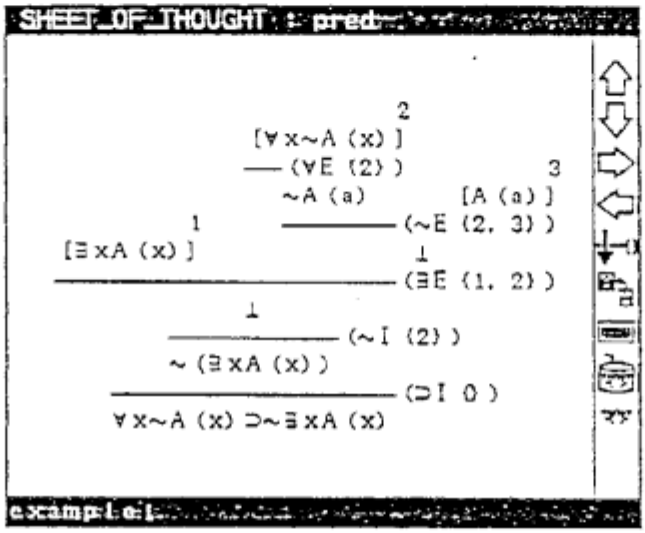

図 6　証明図

Fig. 6 Tree-form Proof

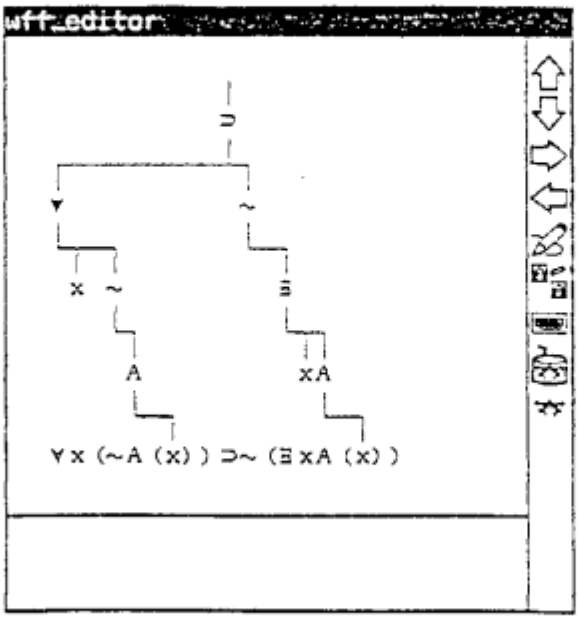

図 7　論理式エディタ

Fig. 7 Well-formed formulas editor

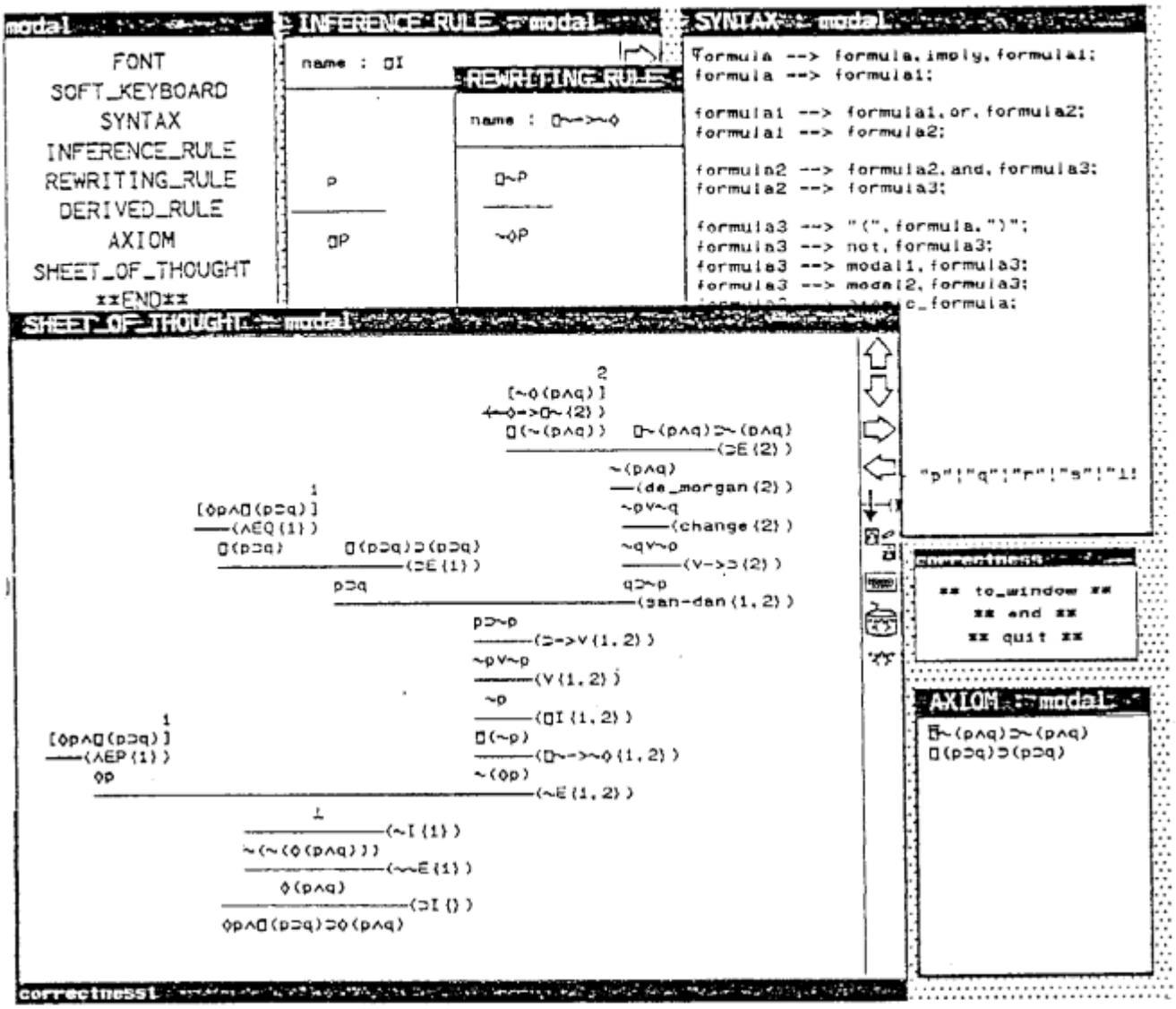

図 8　様相命題論理の定義および証明例

Fig. 8 An example of logic definition and a proof (propositional modal logic)

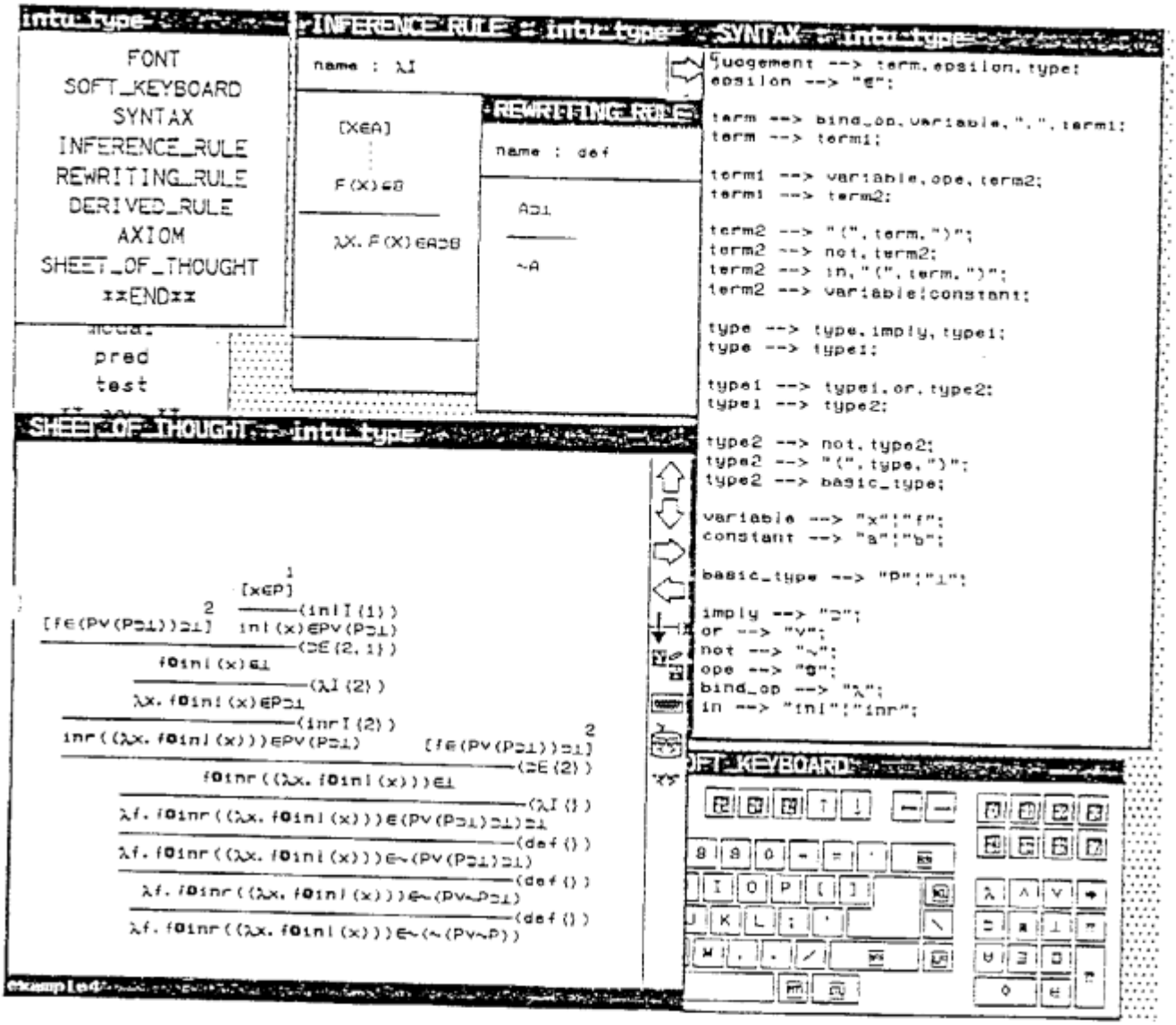

図9　Martin-Löfの直観主義タイプ理論の定義および証明例

Fig.9 An example of logic definition and a proof (Martin-Löf's intuisionistic type theory)