ICOT Technical Report: TR-346

TR-346

論証支援システム：
論理モデル構築のための支援ツール

南　俊郎、沢村　一、佐藤かおる、土屋恭子

March, 1988

©1988, ICOT

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

論証支援システム：論理モデル構築のための支援ツール

南俊朗，沢村一，佐藤かおる，土屋恭子，裵東善

富士通㈱国際研　　㈱富士通研究所

1．まえがき

ある概念を認識し，その概念を何らかの記号系の下で表現（定式化）し，それを用いた形式的な操作によって，その概念に関する考察を行うということは，人間にとって重要な知的活動の1つであると考えられる．形式化された表現を用いることで，自分の得た知見を整理し，厳密な議論を行うことができる．また，客観的な形で他人に伝えることも可能になるという利点もある．我々はこのような，概念の定式化，および，操作という知的活動を計算機によって支援することで，より容易に，概念の形式的表現を行なえるようにしたいと考えている．形式的体系によって概念を表現すること，それを操作することは，それぞれ，一種の論理モデルを定めること，その論理モデルに対する論証を行うことであると考えることができるので，ここで述べたような支援を行うものを「論証支援システム」と呼ぶことにする．

本稿で述べる我々の論証支援システムは，以下のような2つの大きな特徴を持つ：

① 論理系定義機能を用いてユーザが定義した論理系に基づく論証が行える．

② ユーザが普段用いている表現に近い自然な表現を用い，良好な対話型ユーザ・インタフェースによって証明を構成するための思考シートなる支援機能を持つ．

本稿は，論証支援システムの大きな特徴である，これら2点に関する議論を通して，論証支援システムという考え方の全体像を明らかにすることを目的としている．

以下，2章では，定理や証明に関する支援システムの中で論証支援システムがどのような位置を占めるものであるかを考察する．3章では論証支援システムの大きな特徴の1つである論理系定義機能に関して説明を行う．4章では，論証支援システムとユーザとの対話の大部分を担う思考シートのユーザ・インタフェースに関する注意点をユーザ操作に関する議論を通して考察する．5章では，思考シートの持つ証明支援機能について，その特徴的機能について説明を行う．6章では，まとめと今後の展望を述べる．

2．論証支援システムとは

本章では，論証支援システムの特徴の1つである，論理系定義機能の観点からの論証支援システムの位置付けを明らかにしたい．そのために，計算機によって人間の行う論証を支援する手段としてどのようなものが考えられるかを考察し，その考察を通じて論証支援システムがどのように位置付けられるかを説明する．

計算機によって論証を支援するシステムとしては，与えられた定理の証明を探索し，人間に代って証明を行うことを目的とした，定理証明機なる概念がある．しかし，本稿では，人間が証明を行う際の効率を向上させるためのツールとしての計算機による論証の支援システムに的を絞って考察したいため，以下の議論では，定理証明機は除外して考える．

2．1　証明チェッカー

人間の行う論証に対する計算機による支援機能として，まず考えられるのは，ユーザの与えた証明が，正しい推論に基づく証明であるかどうかを検証する機能，すなわち，証明チェッカー（［Constable 82］，［ICOT 86］等）なる概念である．計算機によって証明のチェックを行うためには，まず，何らかの形式的な証明記述言語を設定し，人間の考えた証明をその言語を用いて記述する．その証明記述を証明チェッカーに与え，証明の中で用いられている各推論が形式的に正しく適用されているかをチェックすることで，証明の正しさを確認するわけである．証明チェッカーの概念を図2．1に示す．

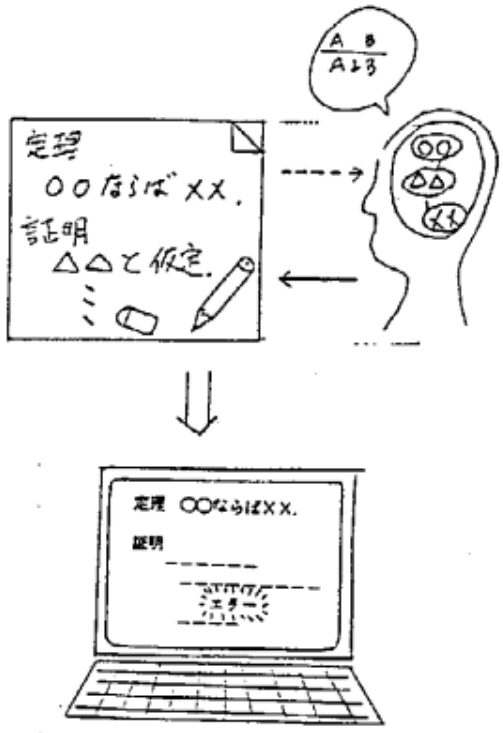


図2．1　証明チェッカーの概念

このようなチェッカーは，ある定理に対する証明が既に

得られており，その証明が本当に正しい証明であるのかどうかを確認したい場合には有用である．しかし，定理を予想し，その証明を構築していく過程というものは，一旦，定理・証明が得られたとしても，その証明を修正し，より完全なものにしあげていく試行錯誤的なものであると考えられる（[Lakatos 76]）ため，これから証明を見付けようというユーザに対する支援ツールとしては，証明チェッカーは役に立たない．

2．2　証明コンストラクタ

試行錯誤的な証明過程の支援を行うためには，証明チェッカー的なアプローチではなく，その代りに，ユーザがコマンドを与え，計算機がその結果を表示するというやりとりを通じて，証明を構築していくという，対話的な証明構築支援機能，すなわち，証明コンストラクタ（証明エディタ）なる概念（[Gordon 79]，[Ketonen 84] 等）が有用である．証明コンストラクタは，通常，ある固定された論理系に対する，専用の証明構築支援機能として実現されている．このように論理系を固定することには，その論理系に依存した最適化の手法の導入が容易であることなどの利点があり，特定の論理系に対する定理・証明構築の支援ツールとして適している．証明コンストラクタの概念を図2．2に示す．

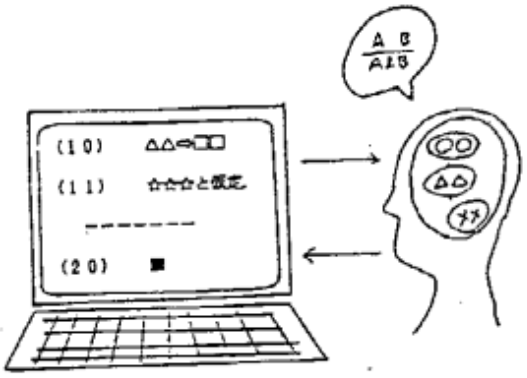


図2．2　証明コンストラクタの概念

2．3　論証支援システム

前節で述べたように，通常の証明コンストラクタは特定の論理系の下での論証を支援するツールとしては有用である．しかし，「全ての議論領域はそれぞれの論理構造を持つ」[Langer 25] という認識に基づいて考えるとき，数多く存在する定式化されるべき問題領域に対する支援ツールとしては問題がある．問題領域毎に，そのための証明コンストラクタを別々に用意するというのでは，無駄が多くて実現可能性に乏しい．また，ある論理系で得られた結果を別の論理系で利用しようというようなことを，一貫性のあるやり方で行うことも難しい．一方，汎用の論理系を定義し，その中で全ての領域を定式化するというアプローチも，実際問題としてそのような論理系を定義できるかどうかに問題がある．また，もし定義できたとしても，そのような論理系に対しては，論理系を固定することによる利点があまり活かせないと考えられる．

このような状況に対処するために，対象とする問題領域毎にそれに適した論理系を定め，（論理系に独立な）証明コンストラクタによって，その論理系の下での証明を構築するという，論証支援システムなる概念を導入する．論証支援システムの下では，様々な論理系が単一の機構で管理され，異なる論理系に対しても同一の手段で取り扱うことが可能となる．

論証支援システムの利用は，まず，対象とする問題領域を記述するための論理系を定義することから始まる．その領域において必要な基礎概念を表現するための記号を予め登録しておく．次に，導入した記号に対してその意図している意味を表現するような公理，領域で成り立つべき推論規則等を記述する．システムはこれらの記述に基づきその論理系の下での証明を支援する．論証支援システムの概念を図2．3に示す．

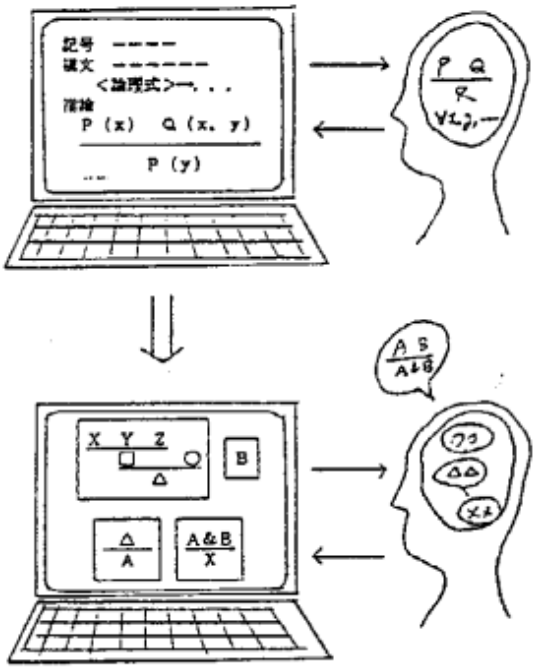


図2．3　論証支援システムの概念

近年，我々の論証支援システムと同様の考え方に基づくシステムがいくらか見受けられるようになった．たとえば，[Griffin 87] にはＥＦＳと呼ばれる論証支援システムの一種が定義されている．そこでは，我々の論証支援システムと同様に構文や推論規則を記述したり，ボトム・アップおよびトップ・ダウンの証明を支援したりの機能が実現されている．しかし，具体的な仕様には違いがある．たとえば，証明の記述法は従来の証明記述言語風であるのに対して，我々は自然演繹法風の記述形態を採っている．また，それに伴い，我々の論証支援システムでは証明スキーマを埋める形式での証明形態が必要となるがＥＦＳでは，そのような形式の証明は考えられていないようである．

本章では，論理系記述機能の観点からの論証支援システムの特徴を明らかにした．次章では，論理系記述機能に関して，より具体的に述べる．一方，論証支援システムが対話型のシステムであることを考えると，ユーザとの対話形式，すなわち，ユーザ・インタフェースへの配慮も非常に重要である．論証支援システムのユーザ・インタフェースに関しては４章，及び５章で詳しく考察する．

３．論理系定義支援機能

前章でも述べたように論証支援システムは論理系定義の可能な証明コンストラクタとして特徴付けることができる．したがって，システムの支援機能は，論理系定義機能と証明支援機能に大別できる．本章では，前者について，具体的に説明する．

論理系定義は，次の３つの定義から成り立つ：

(1) 記号定義

論理系を定義するためには，まず，その論理系でどのような記号をどういう役割の物として用いるかを定めなければならない．システムは記号定義編集機能を提供する．

(2) 構文定義

証明を行う際の基本表現は論理式なる概念である．この論理式なる概念が，どのように表現されるのかの構文規則を与えるのが構文定義の大きな目的である．

(3) 公理・導出規則定義

対象とする論理系の基本的性格を決定するのは，どのような推論形態がその論理系で許されるかの規則である．推論の出発点となる公理，および基本的な推論形式を与えるのが，本定義である．

以下の各節においては，それぞれの定義法について更に詳しく述べる．

３．１　記号定義

記号は論理式等の表現のための基本要素となる文字列である．ユーザはまず，問題領域を記述するために，どのような記号を用いるのかを決定する．どのような記号を定数記号とするか，関数記号とするか，関係記号とするか，論理結合子とするか等を考えるわけである．記号は型，構文型，演算子として用いるかどうかの属性を持つ．もし演算子ならば更にどのようなタイプの演算子であるか等の属性が追加される．記号の属性を与えるとき，個々の記号毎に与える必要はなく，あるパターンの文字列はこのような属性を持つといった形での定義も可能である．記号定義の例を例３．１に示す．

```
a* : t :: constant                                        --①
[i-n ]?* : int :: variable                                --②
f+g+h : int->int :: constant                              --③
"+" : (int,int)->int, <yfx,100> :: constant               --④
"*" : (int,int)->int, <yfx,200> :: constant               --⑤
```

例３．１　記号定義の例

①はａ，ａａ，ａａａ等の名前は全て型ｔの定数であることを宣言している．②はｉからｎで始まる全ての名前はｉｎｔ型の変数であることを示している．③はｆ，ｇ，ｈは全てｉｎｔ型からｉｎｔ型への関数であることを示している．④，⑤はそれぞれ＋，＊が演算子であることを示している．演算子の型は左優先型（ｙｆｘ）であり，その結合力は，それぞれ，１００，２００である．演算子の宣言は次節で説明する構文定義と合せてパーザの生成の際に用いられる．

３．２　構文定義

次に，前節の機能を用いて定義された記号群から，対象を表現するための「項」や，命題を表現するための「（論理）式」などをどのような記号列として組み立てるのかを定めるための構文定義機能について述べる．構文定義の記述法としては，文脈自由文法と類似の記述が可能であり，パラメタを付加することで，もっと一般的な構文をも記述できるという点を考慮して，ＤＣＧ文法（［Pereira 80］）の表現を基本とする．

文脈自由文法記法と類似したＤＣＧ文法と演算子順位文法を比較すると，前者は表現力の点で勝れており，後者は演算子の結合性の理解し易さの点で勝れている．両者の長所を活かすために，本システムではＤＣＧの記述を基礎に，演算子に関しては，その旨の宣言と，その結合力を与えることができることとした．この宣言と結合力は３．１節で説明した記号定義の中で記述される．演算子に関する宣言は，それに相当するＤＣＧ文法の記述を行った場合と同様の解釈でパーザによって利用される．例３．２に演算子宣言とＤＣＧ文法を混用した構文記述の例を示す．

```
記号定義
"+" : (int,int)->int, <yfx,100>  :: constant
"*" : (int,int)->int, <yfx,200>  :: constant
1, 0 : int:: constant

構文定義
<formula> → <Predicate Symbol>,"(",<term-list>,")"
<term-list> → <term>
<term-list> → <term>,",",<term-list>
<term>  → 1
<term>  → 0
<term>  → <term>,"+",<term>
<term>  → <term>,"*",<term>
```

例３．２　構文記述の例

この例では，＋と＊はｔｅｒｍなる構文要素に対する演算子であり，１と０はｔｅｒｍに対する定数であると定義されている．前者の宣言により，システムは，ｔｅｒｍに関して次の構文定義が与えられた場合と同一の処理を行うパーザを生成する．

```
<term>  — <term1>
<term>  — <term> + <term1>
<term1> — <term2>
<term1> — <term1> * <term2>
<term2> — 1
<term2> — 0
```

以上の定義より，'１＋０'，'０＊１＋０'等をｔｅｒｍとして解析できることになる．解析木は通常の数式の場合と同様である．

例３．２からも分るように，記号定義と構文定義は密接に関連しており，構文定義の後に，新しい記号定義を追加することで不足部分を補うようなことも起る．

３．３　公理・導出規則定義

前節までの定義によって，表現のための言語系が設定できたことになる．次に対象領域の論理的構造を表現する段階（導出系定義）へと進む．構造は，公理，および，公理や既に得られている定理から新しい定理を導くための導出規則によって与えられる．

公理は推論の前提となる状態を表現するもので，記述しようとしている領域の持つ基本的性質が記述される．公理は公理定義画面において，順に指定していくことで，システムに入力される．

導出規則には，推論規則と書き換え規則の２つがある．推論規則の形式としては，実際に人間が行う推論の形態により近い形式であるとの認識により，自然演繹法スタイルの推論規則の形式を採用する：

［推論規則の形式］

$$\frac{[\text{仮定式}1]\ [\text{仮定式}2]\ \cdots\ [\text{仮定式}n]}{\text{前提式}1 \quad \text{前提式}2 \quad \cdots \quad \text{前提式}n}$$

結論式

（必要に応じて，固有定数に関する条件を適用条件として付けることができる．また，仮定式の一部，もしくは全部がない場合も許される．）

自然演繹法の場合と同様に，この推論規則は「（仮定式がついている前提式の場合はその仮定式を仮定した上で）全ての前提式が導出されているとき，結論式を導出することができる」という規則を表わしている．

書き換え規則は，次の形式で与えられる：

［書き換え規則の形式］

$$\frac{\text{書き換え前の部分式の形式}}{\text{書き換え後の部分式の形式}}$$

書き換え規則は，「ある式の部分式が書き換え前の部分式の形式と一致する場合，その部分を書き換え後の部分式の形式で置き換えた式を導出することができる」という規則を表している．ある式の部分項をそれと同値な項表現で置き換えるといった操作はこの書き換え規則を用いて行うことができる．

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall x\ y\ z.\ (x+y)*z = x*z + y*z}{(a+0)*b = a*b + 0*b}\ \forall E}{a*b = a*b + 0*b}\ (x+0=x)}{\vdots}}{\forall x.\ 0*x = x}$$

例３．３　推論規則，書き換え規則を用いた導出の例

本章の結果を要約すると，論証支援システムに占める論理系定義支援機能の役割は図３．４に示される．

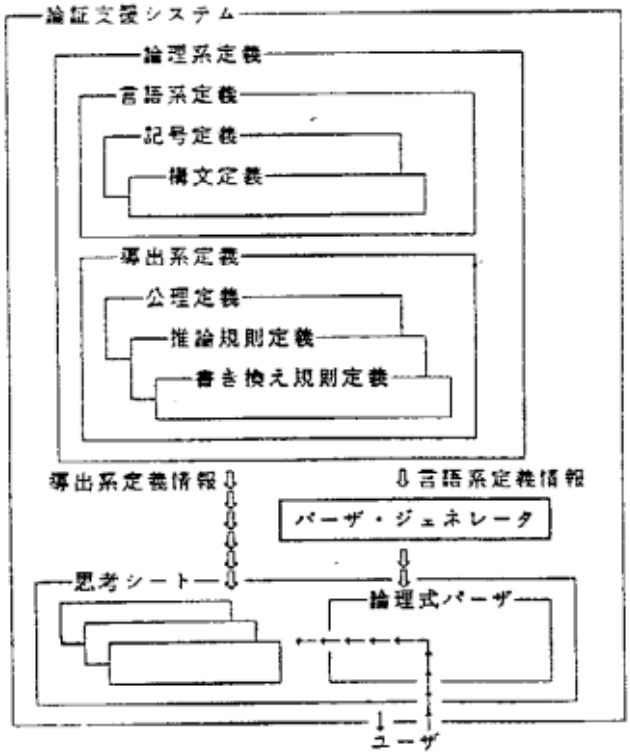


図３．４　論理系定義支援機能

３．４　論理系定義支援機能に関する補足

(1)　前節でも触れたように，本システムの構文定義は，ＤＣＧの表現法によって記述される．通常用いられているＤＣＧパーザはトップ・ダウン・パーザであり，左再帰形での定義式をパーズできない等の問題がある．そのため，我々はそのような問題の起こらないボトム・アップ・パーザであるＢＵＰ（［Matsumoto 83］）を元に，演算子定義を処理するための機能拡張を施したパーザを使用する．

(2)　他のシステムにおいても，記号属性の定義を行うという形式での論理系の拡張が考えられている．たとえば，Ｅ

ＫＬは記号の属性定義という形式で，型，構文型，ソート等を与えることができる．また，演算子に対しては，結合力，結合性等を定めることができる．その他，限量演算子やλの記法を表わすための，束縛演算子型という概念を持っている．しかし，基本的な構文規則は固定されており，論証支援システムのアプローチとは異なる．

(3)　[Harper 87]では，論証支援システムと同様の問題意識より，既存の多くの論理系を包含するような，論理系の一般枠（Logical Framework ）を与えるものとして，型付きのＬＦなる体系を論じている．

4．思考シートのユーザ・インタフェースに関する考察

２章でも分析したように論証支援システムはユーザとの対話を通じて証明を行う．従って，人間の思考にマッチした対話性を持つかどうかは，そのシステムの使い勝手に大きな影響を及ぼす．対話型システムとしての論証支援システムにとって，どのようなユーザ・インタフェースが望ましいのかを考察するためには，システムのユーザがどのような行動をとるのか，また，システムがどのように反応するときにユーザは使い易いと感じるのかを，最初に議論する必要がある．厳密な議論のためには，形式的なアプローチを採らなければならないのであろうが，ここでは，厳密性にはあまりこだわらず，直観的なユーザ像に基づいて考察する．論証支援システムが対象とするユーザとしては，論理に関する知識，経験は持っているが，計算機には必ずしも慣れているとは言えない人を想定することにする．そのような想定に基づき，ユーザの特徴とそれに対してシステムがどう対処すれば良いのかを以下の各節で考察する．

論証支援システムの中でも，思考シートと呼ばれる部分は，ある論理系の下での，定理を予想し，証明を発見するという過程での支援を行うためのツールであるため，人間の発想を妨げない対話性の良さが特に重要である．これまでの知的作業にとって，紙という素材は不可欠のものであった．紙の持つ扱い易さ，柔軟性を保ったまま，計算機上のシステムであることの利点を活かして，紙にはない，高度な支援機能を備えることが論証支援システムの究極の姿であり，思考シートは，その中の「証明構築支援環境」部分である．

本章では，以下の各節でユーザとの対話の様々な側面について議論し，論証支援システムがそれにどう対処するかの例を述べる．まず，４．１節では，ユーザ主導のユーザ・インタフェースに対応する方法について議論する．４．２節では，システムの応答時間に関して配慮すべき点を考察する．４．３節では，計算機システムで有りがちな，使用上の制約について考慮する．最後に４．４節で，ユーザの操作，システムの応答の一貫性について考える．このような考察を通じ，思考シートの備えるべき使い勝手のよいユーザ・インタフェースを議論する．

４．１　ユーザの主導権について

論証支援システムの目的はユーザの仕事が効率良くいくように助けることである．したがって，主導権は常にユーザにある．その考え方に従うと，ユーザは好きな時に好きな行動を開始して良いし，行動の途中，任意の時点で，その行動を取り消すことも許される（「ユーザは王様」という考え方［Hopgood 86］）．

システムはそのような思い付き行動に対しても，ユーザの意図に応えなくてはならない．そのような制約により，システム側が次に入力すべきことを決定し，プロンプトによってユーザに指定するといったインタフェース形式を採ることはできない．ユーザが起した行動をシステムが検出し，それに対する応答を判断するという方式とならざるを得ない．また，ある行動を指示された場合で，それを遂行するには，情報が不足していることがある．そのような場合は，不足している情報を補うための，対話が必要となるが，そのような場合でも，途中いつでも，ユーザは最初に与えた行動の指示の取り消しが行なえるような機能を用意しなくてはならない．

思考シートにおけるこのような配慮の例として，論理式エディタやソフト・キーボードの呼び出しがある．ユーザは論理式を入力する際には，必要を感じた時点で，いつでも，これらを呼び出し，利用することができる．また，必要性を感じないときは，これらを利用せず，直接，キーボードから入力することもできる．

４．２　応答時間の考慮

ユーザが何らかの指示を与えてから，計算機がそれに対する結果を返すまでの応答時間は，対話型システムにおいては重要である．計算機が今どのような処理を行っているのかの予測がつく場合には，それほど，不安を感じないかも知れないが，予測のつかない場合や予測した処理時間を大幅に過ぎても処理が終了しない場合には，大きな不安を引き起すものである．

このような場合においても，ユーザに不安感を感じさせないようにするためには，処理を始める前にシステム側で処理時間の評価を行う必要がある．ある処理が，ユーザが不安を感じずに待つと考えられる一定限度以内の時間で終了することが，予め分っている場合には，特に注意することはない．しかし，長時間かかる処理とか，場合によっては，永久に処理が終らないことが有り得るような処理の場合には，対策が必要となる．まず，処理時間が予測できる場合には，その予測時間を出力する．ユーザはいつまで待てば良いのかが，分るため，その後の作業の計画が立て易い．その処理を対話処理とは分離して，別プロセスとして実行することも考えられる．また，予め処理時間が分らない場合は，処理の途中経過をユーザに知らせることが有効であろう．たとえ，時間がかかろうとも，処理が順調に進んでいるならば，待つことができるからである．システム

側が自ら判断して知らせる他，ユーザが問い合せた時に知らせる機能も必要である．もちろん，ユーザの指示によって，処理を途中で中断・中止する機能も必須である．

我々の思考シートは，現在のところ，基本的な編集機能が主であり，その他の機能も，予想できないような長時間かかる処理を含んでいない．しかし，もっと高度な支援機能を追加する際には，このような配慮が重要となる．

4．3　表現や操作上の制約

使い易いユーザ・インタフェースを目指す立場から，考えると，計算機の上であっても，普段用いている表現を用い，普段経験している操作法で扱える事が望ましい．論証支援システムの対象である論理式の場合で考えると，例えば様相論理の場合には，□，◇等の記号を用いることが可能でなければならない．そのために，論証支援システムは，記号の定義機能，その記号をどのように用いるのかを指定するための構文定義機能などを備える必要があるが，既に，2～3章で述べたように我々の論証支援システムは，記号定義，構文定義機能を用意している．また，5．1節で述べるように，システムとフォント作成機能を結びつけることで，システムの持っていない記号も，そのフォントを定義し，その記号定義，構文定義を追加することで，普通の記号と全く同じに扱うことができる．

4．4　操作・応答の一貫性

ユーザ・インタフェースの一貫性が保たれることの重要性は，論証支援システムに限らず，どのようなシステムに関しても言えることである．ユーザは，画面の表示を通して対象を認識するのであるから，ユーザにとって画面の表示は対象そのものである．同じように見える対象が，同一操作に対して全く異なる応答を示すならば，ユーザは混乱するであろう．したがって，ある対象に対してモードなる概念はできるだけ避けた方が良い．仮に，モードを採用せざるを得ない場合には，何らかの形でそのモードを表示し，そのモードであることが確認できるようにしなくてはならない．5章の図5．4に論証支援システムでのモード表示の例を示す．

一方，対象が異なる場合であっても，同様の処理に対しては同様の操作が対応するならば，既に覚えた操作法を様々な対象に適用できることになり，新たに覚えるべきことが少なくて済む．このような配慮で随分使い易くなるものである（驚き最小の原理［Hopgood 86］）．

このような一貫性を実現するための手法として，「マウスの左ボタンをクリックする」といった，具体的な物理的操作ではなく，「対象を選択する」といった，論理的操作をシステムの基本操作単位とするという処理方法が考えられる．物理的操作と論理的操作の対応付けは，操作の処理部とは別に，用意された対応表を通じて行われる．このような，構成を採ることで，例えば，「対象の選択」の対応付けを変更する場合は，全ての局面における，対象の選択の操作法が一括して変化するわけで，ある部分の選択は「左ボタン」であり，別のある部分の場合は「右ボタンの2回クリック」といったような，操作規約の不整合をなくすることができる．

思考シートにおける例として，たとえば，「元の領域指定」と「先の領域指定」がある．削除を行うためには，「元の領域指定」と削除指定を行えば良いし，ある部分を複写もしくは移動するためには，元と先の両方の指定と共に複写・移動指定を行うと良い．この操作で複写が行われるか移動が行われるかは，複写・移動のどちらのモードであるかにより決る（図5．4参照）．

5．思考シートの機能

既に述べたように，思考シート上の推論は，自然演繹法スタイルで行われる．ユーザは，思考シートの上で推論の木構造を構成したり，削除したり，既存の証明を参照したりしながら，定理の予想，その証明の発見等を行うわけである．推論の形態には，既に得られた結論式，公理，仮定式等から新たな結論式を導出するという，ボトム・アップ方式，逆に，導出したい結論式に対して，それを導くであろう，仮定式を予想し，更にその仮定式を結論式として導くような仮定式を予想していくという過程を繰り返す，トップ・ダウン方式，また，それらを，混合して用いる方式等，様々のパターンが存在する．ユーザは思考シート上で，そのような方式を使い分けながら，種々の証明断片を元に，それらを結合していき，1つの証明（および定理）を作りあげることになる．思考シート上の証明断片の様子を図5．1に示す．

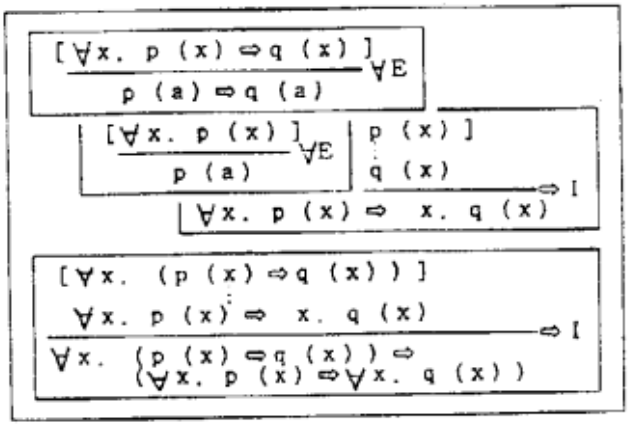


図5．1　思考シート上の証明断片の様子

本章では，思考シートの持つ具体的機能のいくつかについて，以下の各節に分けて説明する．論証支援システムの証明構築支援環境である思考シートの主な側面は，対話型システムであるエディタとしての面である．そのため，思考シートの機能，ユーザ・インタフェースは，前章での考察を参考としなければならない．

5．1　入力支援機能

普通の文書の場合の入力は英字の列であることが多いが，思考シートでは，それと異なり，論理式の入力が主となる．そのため，論理式の正確かつ高速な入力を支援するための機能が有効となる．思考シートは入力の支援ツールとして論理式エディタ（［沢村 86］，［佐原 86］，［土屋 87］）を用意している．このエディタは論理式の木構造を直接操作する，構造エディタであり，基本的な編集機能の他，簡単な書き換え操作が可能である．

一方，論理式の表現には，通常のキーボード上には存在しない，様々な論理記号が用いられるという事情がある．従来のキーボードを活かすために，英字列による名前を論理記号の代りに用いることで，新たな論理記号を定義しないで済ませるという方法も考えられる．しかし，前章で述べたような観点から，ユーザが普段用いている表現にできるだけ近い形式での表現能力を備えようとすると，このような方法は採用できない．

この問題に対処するため，思考シートには，記号を定義するための，フォント・エディタを呼び出す機能がある．また，作成された記号のフォントを入力するための手段を提供するために，画面上に仮想的に作られたソフト・キーボード（仮想キーボード［南 86-2］）と呼ばれるキー入力支援機能も用意されている．フォントは，ソフト・キーボード上で適当なキーに割り当てることができ，その後は，割り当てられたキーに対応する物理的キーを叩くか，マウスで画面上のキーをクリックすることで，その記号を入力することができる．3章で述べた，記号定義，ならびに，構文定義機能を用いることで，その記号を演算子として登録することも自由である．

証明の始まりとなる論理式は，公理等の形態で，その表現が既にシステムに与えられているもの，および，仮定式のように，新たに入力しなければならないものの2種類がある．前者の場合は，メニューを表示し，その中から入力したい式を選択することにすれば良い．後者の場合は，論理式エディタやソフト・キーボード等は任意の時点で呼び出せるため，必要に応じて呼び出し，それらを用いて，新たに入力することになる．

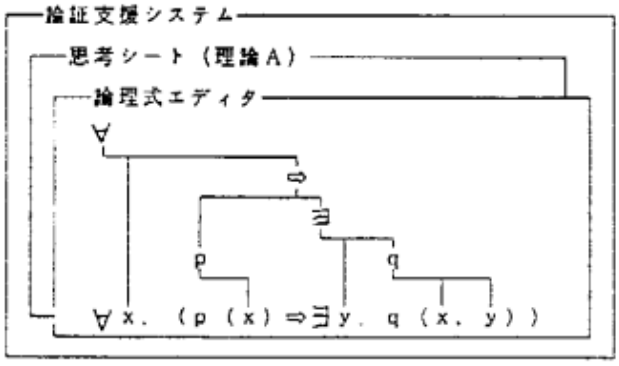
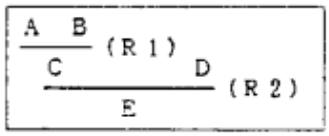


図5．2　論理式エディタの呼び出し

5．2　編集機能

思考シート上の編集機能の概略を述べる前に，証明断片がその上でどのように表現されるのかを示しておく．



この図は式A，B，Dを前提としたとき式Eが導かれることを示している．詳しく述べるならば，式A，Bから導出規則R1を用いて式Cが導かれ，また，導出された式Cと式Dから導出規則R2によって式Eが導かれることをこの証明断片は示しているわけである．

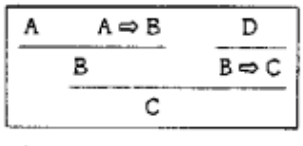
ユーザは思考シート上でこのような形の断片を編集して証明を作り上げるわけである．編集の一般的機能として，削除，置換，複写，移動，探索，操作の取り消し等があるが，証明編集に関しても同様な機能が必要である．その他にも，推論を行う等の編集対象が証明図であることに由来する特殊な機能がある．

(1)　削除機能

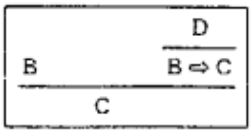
既に存在する論理式や証明断片の一部を削除するためには，削除部分を指定しなければならない．しかし，思考シート上の証明断片の削除は，通常のテキスト編集での削除と異なり，木構造を保つような形で行わなければならないため，幾分複雑になる．たとえば，断片中の任意の論理式だけを削除するといったことはできない．

そのため，思考シート上の削除の指定は，その中に含まれる論理式，および，その結論式方向への削除なのか，もしくは，前提式方向への削除なのかの両方の指定によって行われる．上方（前提式方向）への削除の場合は，指定された式よりも上の部分全てが削除されることになる．下方（結論式方向）への削除の場合は，指定された式を前提式とする導出の結論式，および，その結論式により導出された部分全てが削除されることになる．

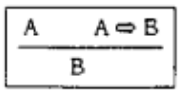
もし，誤った指定をした場合は，取り消し操作によって，元に戻すことが可能である．削除がどのように行われるかを図5．3に示す．

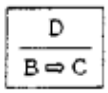


(a)削除前の証明断片



(b)Bの上部分の削除結果





(c)Bの下部分の削除結果

図5．3　削除

(2) 複写・移動機能

複写や移動といった機能は，削除機能と同様の，複写元・移動元の領域指定および，入力の場合と同様の，複写先・移動先の指定との組み合わせで実現できる．思考シートの場合，複写・移動の指定は全く同じ指定法を採用した．それらの区別はモード指定による．現在どのモードであるかが容易に分るように，このモードは常に画面上に表示される．

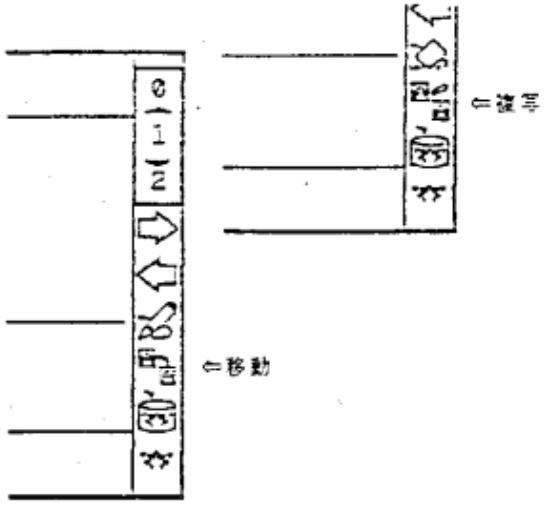
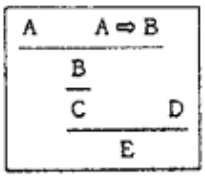


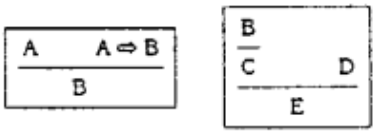
図5．4　複写・移動のモード表示

(3) 分離

木構造特有の配慮が必要ではあるものの，これまで考察してきた機能は一般的な編集機能であった．ここで述べる分離機能は思考シートが木構造エディタであることに依存している．分離機能とは，要するに1つの証明断片を2つの証明断片に分ける機能である．分離に際して，木構造を保つためには，削除機能の説明で考察したのと同様の配慮を行うと良い．すなわち，一般的な削除機能は分離操作の後，分離された証明断片の片方の全体を取り除くことを意味する．



(a)分離前の証明断片

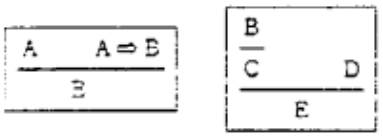


(b) B指定による分離後の証明断片

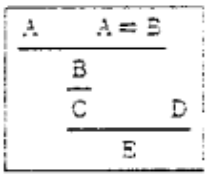
図5．5　分離

(4) 結合

分離の反対の操作が結合である．これは，ある証明断片の結論式と別の証明断片の前提式を同一の式としてつなぎ合せることである．結合の際には，変数名の変更を含む単一化は自動的に行われる．結合の例を図5．6に示す．



(a)結合前の証明断片



(b) Bによる結合後の証明断片

図5．6　結合

5．3　推論規則による証明断片の拡張

一般的な編集機能とは別に証明コンストラクタ特有の機能として，様々な形式で，推論規則を適用し，より大きな推論を生成していく機能がある．本節ではその基本的パターンについて述べる．

(1) ボトム・アップ推論

既に得られている論理式を前提式とし，それらから新たな結論式を導く推論である．

$$\frac{A1,\ldots,An}{}(R) \Rightarrow \frac{A1,\ldots,An}{B}(R)$$

(2) トップ・ダウン推論

ある論理式に対して，それを結論式とするような前提式を導く方式である．これは，①の逆操作である．

$$\frac{}{B}(R) \Rightarrow \frac{A1,\ldots,An}{B}(R)$$

(3) 前提式と結論式より，推論規則を導く

前提式と結論式の双方を与え，適用可能な推論規則を導く方式である．推論規則の合成をも導くのが，いわゆる，自動定理証明と呼ばれる方式である．

$$\frac{A1,\ldots,An}{B} \Rightarrow \frac{A1,\ldots,An}{B}(R)$$

5．4　その他の機能

本節では，編集に直接関わらない，思考シートの機能の幾つかについて説明する．

5．4．1　他理論の参照

ある論理式は，その証明が完成し，その理論の定理とし

て認められたとき，その式とその証明は，現理論に関する定理データベースへ登録され，後の証明の際に公理と同じ扱いで用いられることになる．更に，現在考慮している理論のみではなく，以前に他の理論の結果として得られた定理，および，その証明をも参考にできる場合がある（［南 87-2 ］），ある理論に対して公理や推論規則等を追加してできる理論を元の理論の子理論と呼び，元理論を子理論の親理論と呼ぶことにする．この場合，親理論における定理，および，その証明はそのままで，子理論の定理，証明となる．このような関係にない場合でも，適当な変換を施すことで，ある理論の結果を別の理論で利用できることが多い．このような，関係は理論間関係データベースに格納され，思考シートはこの関係を利用して，現在参照している理論に関する証明を導くことができる．

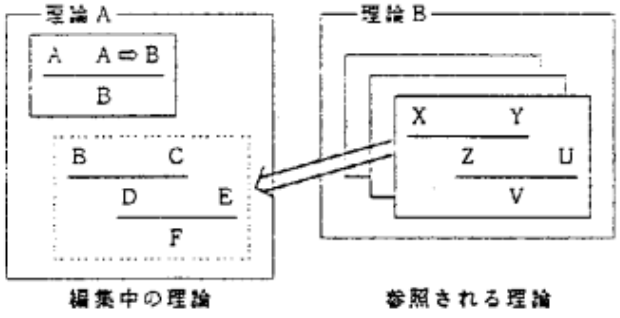


図5．7　他理論の参照

5．4．2　定理・証明の清書機能

前節でも述べたように，一旦得られた定理，および，その証明は，ユーザの指示により，定理データベースへと格納される．この定理，およびその証明を数学で通常用いられる証明形式で清書して出力する機能を思考シートは持っている（［南 87-1 ］）．本機能を用いることで，ユーザは，自然演繹法スタイルの木構造証明図だけでなく，自然言語風の表現の証明を得ることができ，木構造スタイルに慣れていない人に示すことができる．

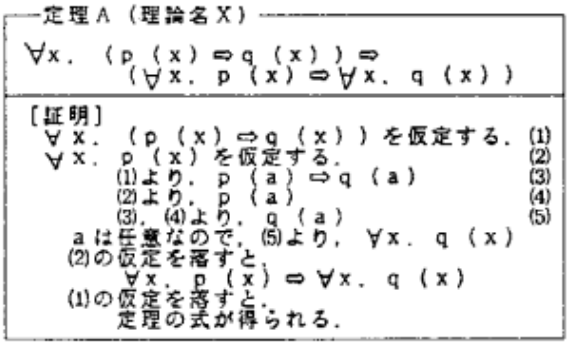


図5．8　定理・証明の清書例

5．4．3　論理系編集

一旦定めた論理系でも，その下での論証を進めていく内に，不備が感じられるようになるものである．その場合には，思考シートの中から論理系編集機能（［土屋 87 ］）を呼び出すことができる．記号の追加，追加された記号に関する構文規則の追加等は，以前の定理・証明に影響を与えないため，自由に行なえる．逆に，記号の削除，公理の変更等の操作は，以前の結果に対する影響がある．そのような，編集は例外的な場合を除き，許されない．

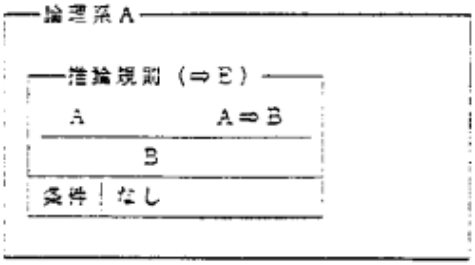


図5．9　論理系編集画面例（推論規則）

5．4．4　派生規則定義

色々な証明を行っていく内に，ある特定の形式の証明が頻繁に用いられることが起る．その場合，その形式の証明を1つの証明パターンとして登録し，その証明形式が派生規則であることの証明を付けることで，その証明パターンは，論理系の持つ推論規則と全く同等に用いることができるようになる．

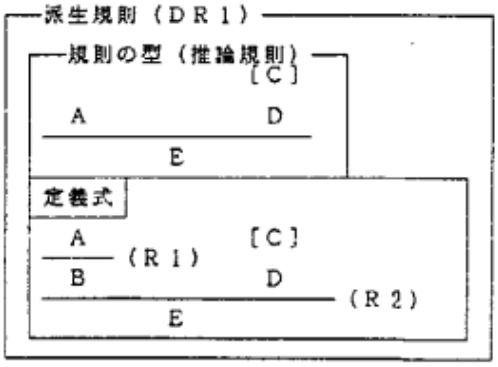


図5．10　派生規則定義

6．むすび

本稿では，論証支援システムの特徴として，次の2点を特に強調した：

①論理系定義機能を持っており，ユーザの定義した論理系に基づいた論証を支援する．

②思考シートと呼ばれる証明構築支援機能を持っている．これは，ユーザにとって，自然な論理式表現，自然演繹法スタイルの（木構造の）証明支援機能を備え，ユーザ・インタフェースに勝れた証明コンストラクタである．

我々は以前，記号の属性定義機能を持ち，EKL風の行エディタ・スタイルで証明を構築するという，証明コンストラクタについて報告を行った（［南 86-1 ］，［南 86-2 ］）．このシステムは，論証支援システムなる概念を明らかにする参考とするために試作されたもので，このシステムの持っている論理系定義機能を拡張し，また，編集ス

タイルを自然な木構造のままで行えるように改良し，発展させたものが本稿の論証支援システムである．現在，これまでの試作の経験を踏まえて，本稿で述べたような，より一般的な論証支援システムの開発を目指して研究を進めている．なお，本論証支援システムの開発言語は推論機械ＰＳＩ上のＥＳＰである．

十分な一般性を持ち，良好なユーザ・インタフェースを備えた論証支援システムの実現のためには，本稿で議論された事柄以外にも，理論間の関係の管理に関する問題（[岡 87-2]）を初め，ＬＣＦ／ＭＬ（[Gordon 79]），ＦＯＬ（[Weyhrauch 80]）に見られるようなメタ理論の導入，簡単な証明を自動的に行ってくれる定理証明機能，証明の変換機能，定理予想機能等，について，今後考察を進めたいと考えている．

謝辞

日頃，御指導，御鞭撻を頂く国際研の北川敏男会長，並びに渡本室所長に感謝いたします．なお，本研究の一部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

[Aponte 84] M. V. Aponte, J. A. Fernandes & P. Roussel: Editing First-Order Proofs: Programmed Rules vs. Derived Rules, Int. Symp. on Logic Programming, pp. 92-98, 1984.

[Constable 82] R. L. Constable, S. D. Johnson & C. D. Eichenlaub: An Introduction to the PL/CV2 Programming Logics, LNCS 135, Springer-Verlag, 1982.

[Eriksson 82] A. Eriksson, A.-L. Johansson & S.-A. Tarnlund: Towards a Derivation Editor, Proc. of the 1st Int. Logic Programming Conf., pp. 146-156, 1982.

[Gordon 79] M. J. Gordon, A. J. Milner & C. P. Wadsworth: Edinburgh LCF, LNCS 78, Springer-Verlag, 1979.

[Goguen 83] J. A. Goguen & R. M. Burstall : Introducing Institutions, LNCS 164, Springer-Verlag, pp. 221-570, 1983.

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

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

[Hopgood 86] F. R. A. Hopgood, D. A. Duce, E. V. C. Fielding, K. Robinson & A. S. Williams (eds) : Methodology of Window Management, Springer-Verlag, 1986.

[ICOT 86] 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．

[Ketonen 84] J. Ketonen & J. S. Weening: EKL - An Interactive Proof Checker, User's Reference Manual, Dept. of Computer Science, Stanford University, 1984.

[Lakatos 76] I. Lakatos: Proof and Refutations, Cambridge University Press, 1976.（佐々木力訳：数学的発見の論理，共立出版，昭和６０年．）

[Langer 25] Langer, S. K.: A set of postulates for the logical structure of music, Monist 35, pp. 561-570, 1925.

[Matsumoto 83] Y. Matsumoto, H. Tanaka, H. Hirakawa, H. Miyoshi & H. Yasukawa : BUP : A Bottom-Up Parser Embedded in Prolog, New Generation Computing, Vol. 1, pp. 145-158, 1983.

[Pereira 80] F. C. N. Pereira & D. H. D. Warren : 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.

[Weyhrauch 80] R. W. Weyhrauch: Prolegomena to a Theory of Mechanized Formal Reasoning, AI Jounal 13, pp. 133-179, 1980.

[沢村 86] 沢村，南，佐藤，小野，小野：論理式エディタ，構造エディタに関するワークショップ，１９８６年５月．

[佐藤 86] 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年９月．

[南 86-1] 南，沢村：論証支援のための証明コンストラクタ，情報処理学会第３３回全国大会，１９８６年９月．

[南 86-2] 南，沢村：論証支援システムの一構成，日本ソフトウェア科学会第３回大会，１９８６年１０月．

[土屋 87] 土屋，佐藤他：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３５回全国大会，１９８７年９月．

[南 87-1] 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年９月．

[沢村 87] 沢村，南：汎用の論証支援システムの構想とその実現法，情報処理学会ソフトウェア基礎論研究会，１９８７年１０月．

[南 87-2] 南，沢村：論証支援システムにおける理論間の関係構造，情報とサイバネティックス・シンポジウム，１９８７年１０月．