ICOT Technical Report: TR-171

TR-171

論理式エディタ
—論証支援システムのための論理式作成ツール—

南　俊朗、沢村　一、佐藤かおる
小野美由紀、小野越夫
（富士通）

April. 1986

©1986, 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

論理式エディタ

—— 論証支援システムのための論理式作成ツール ——

南　俊朗，沢村　一，佐藤かおる，小野美由紀，小野越夫
（富士通・国際研）　　（富士通研究所・ソフト研）

1．はじめに

我々は現在，汎用の論証支援システムを研究しているが，これはユーザが記号体系，推論規則等によって論理系を与えると，その論理系に基づいた論証を行う際，会話的に支援するものである。このようなシステムの場合，論証支援機能と共にどのようなユーザ・インターフェース機能が提供されるかがユーザの使い勝手にとって重要な要素であると考えられる。たとえば，論証支援システムの使用に際して，ユーザは様々な論理式を入力しなければならず入力の容易さを向上させるための工夫が必要である。論理式エディタ機能は論理式入力のためのフロント・エンド機能の１つと考えることにする。すなわち，論証支援システムにおける論理式エディタの位置付けは図１のようになる。

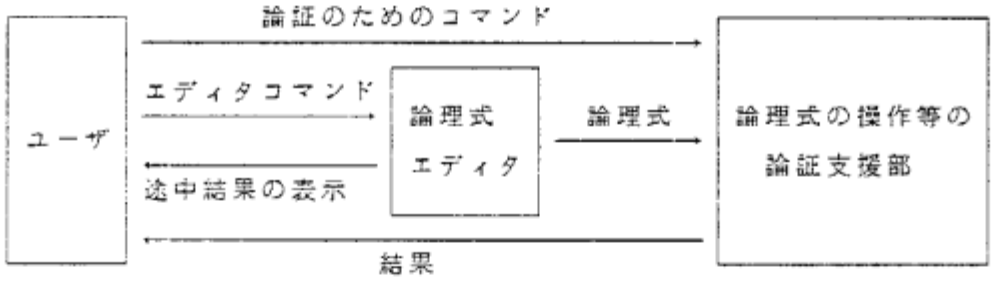

図１　論理式エディタの位置付け

すなわち，ユーザはシステムの操作部に直接論理式を与える代りに論理式エディタによって希望の論理式を会話的に作り上げ，できあがった式を操作部に引き渡すわけである。

このような論理式エディタを介在させることで次のような利点が予想される。

① タイプ・ミスによって，折角途中まで入力した論理式を再度打ち直さないで良くなる。部分的に入れておいて後で修正を施せば良い。

② 以前に作成した論理式を呼び出し，それに加工を施すことで，それほどの手間をかけずに僅かの修正で必要な論理式が作成できる。また幾つかの論理式を組み合せて新しい論理式を作り上げることも容易に行なえる。

なお，論証支援システム全体としては証明・推論に関するエディタ機能，すなわち，入力された論理式に推論を施して証明を作り上げるための機能，も必要となる。本システムでは論理式操作のための論証支援部の一部として実現見込みである。既に発表されている

証明・推論エディタとしては，［1］，［2］，［3］等がある。

今回報告するシステムは上記の論理式エディタがエディタとしてどのような機能を備えているべきであるかを検討する際の参考とするために試作したものあり，それだけで1つの論理式エディタとして使用されることを想定している。使用言語は FACOM OSIV/F4下の M-Prolog である。また，本エディタにおいて論理式は構文的には Prolog の項であるため，論理式に対する固有のコマンドを使用しない限り Prolog エディタとしての利用も可能である。使用例を付録に示す。

2．論理式エディタの概要

本エディタはユーザが定義した論理系に基づき，その下での論理式の編集を行うためのシステムである。編集対象となる論理式は適当な述語名を持つ述語の引数とした Prolog の項として assert されているものとする。編集述語 edtに対してこの述語名を指定することにより，本体部分の論理式の編集を行なえる。たとえば，式

```
all(x, x*1=x)
```

は，述語 p をつけて

```
p(all(x, x*1=x))
```

と assert しておけば，

```
edt(p)
```

という述語呼び出しによって，編集できる。

エディタの処理の本体部は，次に示す2つのステップの繰り返しである。

ステップ1　コマンドを入力する。
ステップ2　コマンドに応じた処理を行う。

ただし，終了コマンドの場合にはステップ2の後，エディタ全体が終了するためステップ1には戻らない。

システムの全体は図2に示す構成となっている。

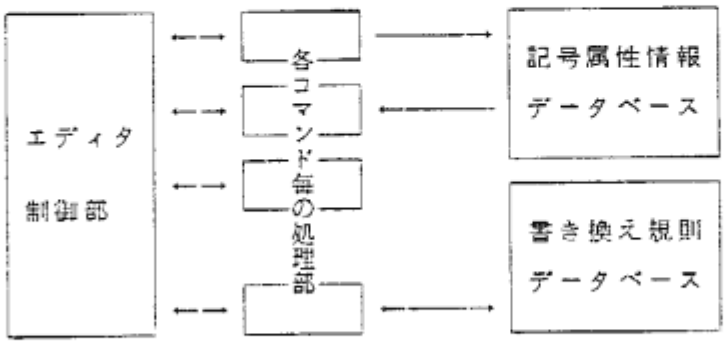

図2　論理式エディタの構成

コマンド全体は，その機能により大きく３つに分けることができる。

(1)　木構造操作用コマンド

論理式は，部分式を部分木とする木構造になっており，編集機能としては木構造エディタ機能が主となる。例えば，論理式の木構造を表示する機能，部分式の追加，削除，置き換え等の編集機能がある。

(2)　論理式操作用コマンド

単なる木構造を操作する以外に論理式であることを考慮した機能がいくつかある。例えば，論理式で用いられる記号の属性の定義，その表示，式の変形，型推論等の機能が実現されている。

(3)　その他のコマンド

(1)，(2)以外にエディタを終了させるコマンド，直前のコマンドを再実行したりある時点以降の修正結果を無効にし，その時の状態に戻すコマンド等がある。

各コマンドの機能に関しては以下３節～５節でより詳しく説明する。

３．木構造操作用コマンド

Prolog 項（特に論理式）は，部分項を部分木とする木構造を持つ。本エディタにおいては，ある部分項を指定して，そこにある操作を加えるというのが基本操作となる。display コマンドによって論理式の全体，もしくは，指定された部分項を表示できる。本エディタでは式としての見易さを考え，通常の木表示と異なり，部分木の各ノードに，対応する部分項全体を付けて表示することにした。次に示す図３がその例である。

```
1  all(x,x+1=add1(x))
|
|----- 1-1 x
|
|----- 1-2 x+1=add1(x)
     |
     |----- 1-2-1 x+1
     |      |
     |      |----- 1-2-1-1 x
     |      |
     |      |----- 1-2-1-2 1
     |
     |----- 1-2-2 add1(x)
            |
            |----- 1-2-2-1 x
```

図３　display コマンドによる式 all(x,x+1=add1(x)) の表示例

部分項の表示の前の，ハイフンで区切った数字の並び，例えば＂１－２－１＂はノード・ナンバと呼ばれ，その部分項の式全体における相対的な位置を表している。以下，各種のコマンドにおいて部分項そのもの，もしくはそのノード・ナンバによって特定の部分項が指定される。部分項指定で候補ノードが複数個存在しうるコマンドの場合，対象となる

部分項が正しいかどうかの確認が行われる。

木構造を変えるコマンドには replace（置換），exchange（交換），insert（挿入；insert_a：直後への挿入，insert_b：直前への挿入），delete（削除），move（移動；move_a：直後への移動，move_b：直前への移動），copy（複写；copy_a：直後への複写，copy_b：直前への複写）といった通常の編集機能が含まれている。

次は move_a コマンドの例である。

［例］（式 f(a,b)=g(c) を f(a)=g(c,b) に変える）

```
(1) ed-wff>
display.
1  f(a,b)=g(c)
|
|----- 1-1  f(a,b)
|     |
|     |----- 1-1-1  a
|     |
|     |----- 1-1-2  b
|
|----- 1-2  g(c)
      |
      |----- 1-2-1  c
(1) ed-wff>
move_a(1-2-1,1-1-2).
f(a)=g(c,b)  b moved to <1-2-2>
```

4．論理式操作用コマンド

論理式特有の処理を行うために，記号の属性の定義・表示，論理系に依存した式の変形機能等を持つ。記号の属性としては variable コマンドによって変数もしくは変数の一般形及び型が定義でき，同様に const，function コマンドで，それぞれ定数，関数を定義できる。例えば＂ｘで始まる名前は変数とする＂場合

```
variable(x_)
```

と宣言するとよい。下線（_）は任意の文字列を表す。

演算子の定義としては logical_op，op_dcl によってそれぞれ論理演算子，一般の演算子の宣言を行うことができる。例えば

```
logical_op(120,yfy,and)
op_dcl(100,xfy,//)
```

によってｙｆｙタイプの論理演算子＂ａｎｄ＂及びｘｆｙタイプの演算子＂／／＂が定義される。ここにｙｆｙタイプは通常の Prolog には無いタイプであり，結合律を満たす演算子であることを示す。すなわち，上記の宣言の後

（Ａ　ａｎｄ　Ｂ）ａｎｄ　Ｃ

Ａ　ａｎｄ　（Ｂ　ａｎｄ　Ｃ）

は共に論理式Ａ，Ｂ，Ｃの連言「Ａ　ａｎｄ　Ｂ　ａｎｄ　Ｃ」を表すものと解釈される。演算子の定義時に型指定を追加することもできる。ただし logical_op による宣言の場合は fx,fy,xf,yf型の場合は bool->bool 型，それ以外の場合は (bool,bool)->bool型とし

て宣言されたものと解釈される。

定義された記号定義を表示させるためのコマンドが symbol_information であり，引数なしの場合は現在の記号の定義状況が表示され，引数として記号名，ノード・ナンバを指定することでそれぞれ記号名についての情報，そのノード・ナンバを持つ項に対する記号情報を表示させることができる。またノード・ナンバ指定の時は指定項に対する型推論が行われ，項の内部間及び項の内部と型宣言間で矛盾が生ずる時は警告メッセージが出力される。

論理式固有の変形演算として subst（代入），simplify（簡約化）の２つのコマンドがある。代入コマンドは代入リストを引数として与えることで変数への代入を行う。例えば，式

（ｘ＋１）＊（ｘ－１）＝ｘ＊＊２－１

を代入コマンド

subst(a+b/x)

によって変形すると次の式が得られる。

（（ａ＋ｂ）＋１）＊（（ａ＋ｂ）－１）＝（ａ＋ｂ）＊＊２－１

ここに，ｘは変数であるものとする。

簡約化コマンドは rule コマンドによって与えられている書き換え規則に従って式を書き換える。例えば書き換え規則

（ｘ＋ｙ）＊＊２　＝　ｘ＊＊２＋２＊ｘ＊ｙ＋ｙ＊＊２

を上式に適用すると次の式が得られる。

（（ａ＋ｂ）＋１）＊（（ａ＋ｂ）－１）＝ａ＊＊２＋２＊ａ＊ｂ＋ｂ＊＊２－１

ルールに関する情報は rule_information コマンドによって表示することができる。

５．その他の機能

これまで説明を行った機能の他，ユーザの使い勝手を向上させるための機能として redo，undo，locate　がある。

redoは直前のコマンドを再実行するためのもので，同じコマンドを再び入力する手間を省ける。

undoコマンドによって直前のコマンドによる式の変更結果を取り消し，もとの式に戻すことや以前の任意の時点の式に戻すことが可能である。本エディタのプロンプト・メッセージは，例えば

(12) ed-wff>

に見られる＂１２＂のようにプロンプト・ナンバと呼ばれる数字が付いている。undoコマンドの呼び出しの際，引数として以前現れたプロンプト・ナンバを指定すると，その時点の状態に戻すことができる。

locateコマンドによってある特定の部分木に位置付けを行う。以後部分木のノード指定として"＊"を用いることで位置付けられた部分木を参照することができる。エディタを終了させるためには end, quitの２つのコマンドを用いる。前者は現在得られた論理式を保存したのち終了するものであり，後者は編集された論理式を保存せずエディタ起動以前の状態に戻して終了するものである。

6．まとめ

論証支援システムの論理式入力部としての論理式エディタのプロトタイプの紹介を行った。現在の所，論理式エディタ部分のみを独立させて試作し，使用経験に基づいて改良を行っている段階である。現在のエディタの機能は木構造エディタ機能，論理式用機能，その他の機能の３つに大別できる。木構造エディタ機能としては，木構造表示，基本的な木構造編集機能を備えている。本エディタで扱う論理式は Prolog の項形式であるので Prologエディタとしての利用も可能である。論理式エディタとしての固有機能として記号の定義及び情報表示，操作のための代入，簡約化の機能を持つ。簡約化のための規則はユーザが与える。

本エディタの今後の課題としては，コマンドの追加・拡張，とくに論理式固有機能の追加が重要である。またユーザの使い勝手の上からはスクリーン・エディタ化他使用経験を活かした改良が必要となる。

謝辞

日頃御指導，御鞭撻をいただく国際研の北川会長，及び榎本所長に感謝いたします。なお，本研究の一部は第５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] M.V. Aponte, J.A. Fernandez & P. Roussel : Editing First-Order Proofs: Programmed Rules vs. Derived Rules, Int. Symp. on Logic Programming, 1984.

[2] A. Eriksson, A.-L. Johansson & S.-A. Tarnlund : Towards a Derivation Editor, Proc. of the 1st Int. Logic Programming Conf., 1982, Marseille, France.

[3] 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.

付録（使用例）

【プログラムの性質の証明の例】

［書き換え規則を作る］

```
edt(rule).
edit
  1  app(0,u)=u
  2  app(x,y,u)=x,app(y,u)
  3  x,app(x,y,u)=app(x,y,u)
  4  u=app(0,u)

*** enter editor commands ***
(1)ed-wff>
insert_h(4,app(app(xy),z)=app(x,app(y,z))).

1   app(0,u)=u
2   app(x,y,u)=x,app(y,u)
3   x,app(y,u)=app(x,y,u)
4   app(app(xy),z)=app(x,app(y,z))
5   u=app(0,u)
app(app(xy),z)=app(x,app(y,z))  <4>    inserted.
(2)ed-wff>
display(4).

"4" is node_number.
OK(y/n)?
y

4  app(app(xy),z)=app(x,app(y,z))
|
|----- 4-1  app(app(xy),z)
|     |
|     |----- 4-1-1  app(xy)
|     |     |
|     |     |----- 4-1-1-1  xy
|     |
|     |----- 4-1-2  z
|
|----- 4-2  app(x,app(y,z))
      |
      |----- 4-2-1  x
      |
      |----- 4-2-2  app(y,z)
            |
            |----- 4-2-2-1  y
            |
            |----- 4-2-2-2  z
(2)ed-wff>
replace(4-1-1-1,x).

1   app(0,u)=u
2   app(x,y,u)=x,app(y,u)
3   x,app(y,u)=app(x,y,u)
4   app(app(x),z)=app(x,app(y,z))
5   u=app(0,u)
   xy <4-1-1-1> replaced to x
(3)ed-wff>
insert_a(4-1-1-1,y).

1   app(0,u)=u
2   app(x,y,u)=x,app(y,u)
3   x,app(y,u)=app(x,y,u)
```

```
4    app(app(x,y),z)=app(x,app(y,z))
5    u=app(0,u)
   y <4-1-1-2>      inserted
(4)ed-wff>
end.

saved:
  rule(app(0,u)=u).
  rule(app(x,y,u)=x,app(y,u)).
  rule(x,app(y,u)=app(x,y,u)).
  rule(app(app(x,y),z)=app(x,app(y,z))).
  rule(u=app(0,u)).

Do you save the definition ? (y/n)
y

end
```

［任意のx,y,z に対して app(app(x,y),z)=app(x,app(y,z))　が成り立つことの証明］

```
edt(example).

edit
  1  app(app(0,y),z)=app(app(0,y),z)
  2  v=v

*** enter editor commands ***
(1)ed-wff>
variable(u,v,x,y,z).

variable type  u
variable type  v
variable type  x
variable type  y
variable type  z    defined
(2)ed-wff>
subst(app(app(x,y,u),z)/v).

1    app(app(0,y),z)=app(app(0,y),z)
2    app(app(x,y,u),z)=app(app(x,y,u),z)
   app(app(x,y,u),z) substituted for  v
(3)ed-wff>
simplify(1-2-1).

1    app(app(0,y),z)=app(y,z)
2    app(app(x,y,u),z)=app(app(x,y,u),z)
   <1-2-1> simplified
   rule : app(0,u)=u
OK(y/n)?
y

(4)ed-wff>
simplify(1-2).

1    app(app(0,y),z)=app(0,app(y,z))
2    app(app(x,y,u),z)=app(app(x,y,u),z)
   <1-2> simplified
   rule : u=app(0,u)
OK(y/n)?
y

(5)ed-wff>
simplify(2-2-1).

1    app(app(0,y),z)=app(0,app(y,z))
2    app(app(x,y,u),z)=app(x,app(y,u),z)
   <2-2-1> simplified
   rule : app(x,y,u)=x,app(y,u)
OK(y/n)?
y
```

```
(6)ed-wff>
simplify(2-2).

1    app(app(0,y),z)=app(0,app(y,z))
2    app(app(x,y,u),z)=x,app(app(y,u),z)
   <2-2> simplified
   rule : app(x,y,u)=x,app(y,u)
OK(y/n)?
y

(7)ed-wff>
simplify(2-2-2).

1    app(app(0,y),z)=app(0,app(y,z))
2    app(app(x,y,u),z)=x,app(y,app(u,z))
   <2-2-2> simplified
   rule : app(app(x,y),z)=app(x,app(y,z))
OK(y/n)?
y

(8)ed-wff>
simplify(2-2).

1    app(app(0,y),z)=app(0,app(y,z))
2    app(app(x,y,u),z)=app(x,y,app(u,z))
   <2-2> simplified
   rule : x,app(y,u)=app(x,y,u)
OK(y/n)?
y

(9)ed-wff>
quit.

example is not changed.
  example(u=u).
  example(v=v).
quit
```

【仕様からHornプログラムを導出した例】

以下に、現在の論理式エディタを仕様からのホーン節プログラムの導出に用いた例を示す。導出するプログラムは部分集合プログラムで、記法は分かり易さのために通常の記法を用い、式の簡約のプロセスのみを示す。

```
rule(z∈nil = false).

rule( (false  → p) = true).

rule( ∀z true = true).

rule(z∈u.v = (z=u or z ∈ v)).

rule((p or q) →  r = (p→r) and (q → r)).

rule( ∀z(p and q) = (∀z p) and (∀z q)).

rule( ∀z (z=u→ z∈y) = u∈y).

rule( ∀z (z∈ v→ z∈y) = v⊆y).

rule((p → true) = p).
```

$x \subseteq y \leftarrow \forall z\,[\, z \in x \rightarrow z \in y\,]$

① subst(nil/x)

$nil \subseteq y \leftarrow \forall z\,[\, \underline{z \in nil} \rightarrow z \in y\,]$

$nil \subseteq y \leftarrow \forall z\,[\, \underline{false \rightarrow z \in y}\,]$

$nil \subseteq y \leftarrow \underline{\forall z\ true}$

$nil \subseteq y \leftarrow \underline{true}$

$nil \subseteq y$

② subst(u.v/x)

$u.v \subseteq y \leftarrow \forall z\,[\, \underline{z \in u.v} \rightarrow z \in y\,]$

$u.v \subseteq y \leftarrow \forall z\,[\, \underline{(z=u \text{ or } z \in v) \rightarrow z \in y}\,]$

$u.v \subseteq y \leftarrow \underline{\forall z\,[(z=u \rightarrow z \in y) \text{ and } (z \in v \rightarrow z \in y)]}$

$u.v \subseteq y \leftarrow \underline{\forall z\,(z=u \rightarrow z \in y)} \text{ and } \forall z\,(z \in v \rightarrow z \in y)]$

$u.v \subseteq y \leftarrow u \in y \text{ and } \underline{\forall z\,(z \in v \rightarrow z \in v)]}$

$u.v \subseteq y \leftarrow u \in y \text{ and } v \subseteq y$