TR-416

論証支援システムEUODHILOSにおける論理式の構文記述法とパーサ生成

南　俊朗、沢村　一(富士通)

July. 1988

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EUODHILOSにおける論理式の構文記述法とパーサ生成

Syntax Description of Logical Formulas and Parser Generation in the Reasoning Assistant System EUODHILOS

南 俊朗, 沢村 一
Toshiro MINAMI, Hajime SAWAMURA
tos@iias.fujitsu.junet, hajime@iias.fujitsu.junet

富士通（株）国際情報社会科学研究所
410-03 沼津市宮本１４０
International Institute for Advanced Study of Social Information Science (IIAS-SIS),
FUJITSU LIMITED, 140 Miyamoto, Numazu, Shizuoka 410-03, Japan

要約

論証支援システムEUODHILOSにおいて論理式の構文がどの様に記述されるのか、またユーザが定義した論理式に対してどの様なパーサが生成されるのかを紹介する。まず、論証支援システムとはどの様な考え方に基づいているかを述べ、次に、そのようなシステムを目指して開発中のEUODHILOSについて、その特徴を概説する。このような背景の下で、論理式構文の記述法とそのパーサについての現状と問題点を述べる。

Abstract

In the reasoning assistant system EUODHILOS, the syntax of logical formulas is given by the user. This paper describes how the syntax is described and also how the parser for it is generated. First, the underlying philosophy of the system is discussed. Next, some of the features of EUODHILOS are presented. Finally, current situations and future problems of the description language for logical formulas and of the parser are mentioned.

1. 論証支援システム EUODHILOS とは何か

本節では、論証支援システム EUODHILOS について、その基礎となる考え方およびシステムの概要を述べる。まず、1.1 節で論証支援システムとはどの様な性格のシステムであるかを説明する。次に、1.2 節において、現在開発中の EUODHILOS について、その構成を具体的に説明する。

1.1 論証支援システムとは

我々がある何物かを議論の対象として認め、それに関して意思疎通を図ろうというときには、その対象を表現する手段、すなわち「言語」がまず必要である。次に、その対象を形式的にどう捉えるかを明らかにすることでより客観的な議論が可能となる。ある対象がどのようなものであるかを形式的に表現することは、その対象の論理的構造を記述することと解釈することができる。本稿では、このような考え方で論理というものを捉え、対象の論理的構造に関する議論を行うことを「論証」と呼ぶ。

このような論証活動はいろいろな場面で見ることができる。例えば、数学的構造は、上記の意味での論理的構造の例であり、したがって数学的構造を操作することは論証である。また、プログラミングの過程において、対象を表わすためのデータ構造を考え、その構造に基づいて対象に関する性質を調べる述語や対象を操作するオペレータを記述する活動も論証であるといえよう。そのほかの分野においても、何らかの対象の持つ構造を明らかにしようという場合には論証が必要とされる。

人間の行うこのような論証を計算機で支援することで論証の効率化を図ることが論証支援システムの研究目的である。上記の定義に依り、論証においては論理そのものを構築するため、あらかじめ論理系を固定して考えるわけには行かず、一種の汎用性が必要である。ここでもう 1 点注意が必要なのは、ある対象の定式化のためには試行錯誤を繰り返さなければならないことである。人間の対象の捉え方は、直観的な場合が多く、直観から直ちに厳密な定式化を導くことは困難であり、試行錯誤を経て初めて完成する性質のものである。([Lakatos 76]) したがって、論証支援システムは人間の試行錯誤過程を支援するものでなければならず、そのための配慮が要求される。

以上の考察より、論証支援システムを特徴付ける機能として、特に次の 2 点を重要なものと考えることができる。

(i) ユーザは記述に必要な論理系を自ら定義し、システムはそれに基づいた論理系を取り扱う機能を持つ。

すなわち、論証支援システムは、特別な論理系に依存しない（論理系独立な）システムである。

(ii) 対話型の証明構成支援機能を持つ。

すなわち、あらかじめ考えられた証明を後でチェックするのではなく、ユーザには、システムとの対話を通じて定理や、その証明を試行錯誤的に発見するといった支援環境が提供される。

我々が開発中の論証支援システムの名前である“EUODHILOS”は、これらの特徴の内、特に（i）の部分を言い表わしている次の文に由来している：

全ての議論領界はその論理的構造をもつ。

“Every universe of discourse has its logical structure.” [Langer 25]

以上のような特徴を持っている論証支援システムなる概念をまとめると図1に示されるような使われ方をするシステムであるといえる。

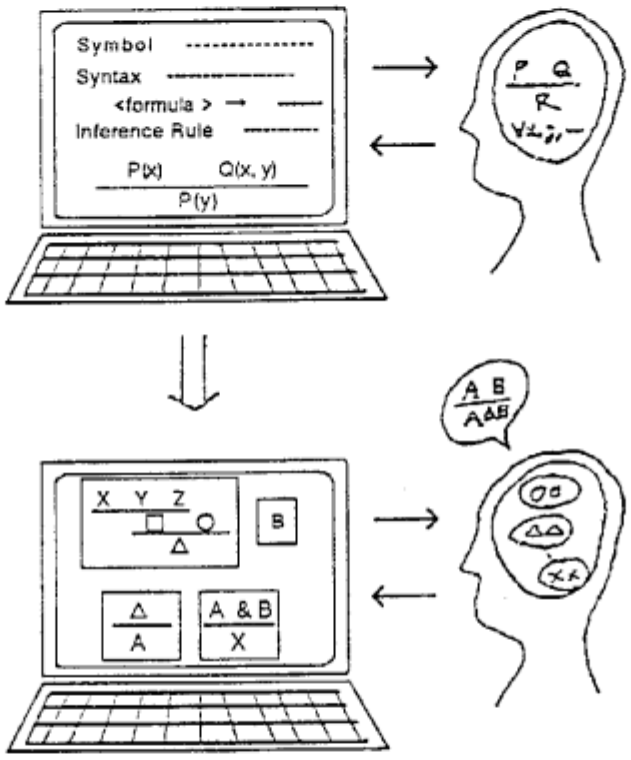

図1 論証支援システムの概念

従来から論証に関連したシステムの形態として、定理の自動証明機、証明チェッカ、証明エディタ等が存在している。定理証明機は自動的に定理の証明を見つけようという趣旨のプログラムであり、この点でそのほかとは異なっている。少なくとも現在の段階では実用的に役立つほど複雑な定理の自動証明は困難であると考えられるため、以下の考察からは除外する。しかし、ある程度の自動証明機能は、そのほかのどのシステムにおいても有用であり、そのようなシステムの部分機能としては重要である。

証明チェッカを、人間が何らかの証明記述言語で、ある定理の証明を記述し、その結果をシステムにかけてその正当性をチェックする形態で論証を支援するシステムであると定義することにする。このような意味での証明チェッカとしてはAUTOMATH([de Bruijn 70]), PL/CV2([Constable 82]), CAP-LA([ICOT 86]) 等がある。扱う論理系としては、ある特定の論理系を仮定するものが普通であるが、我々の論証支援システムと同様に論理系に

依存しないもの（例えば、AUTOMATH）も存在する。この形態のシステムは人間が既に持っている証明の正当性を確認する場合には適しているものの、これから試行錯誤的に定理や、その証明を見つけようという目的にはあまり適さない。

そのような目的のためには証明エディタ的アプローチがより適している。証明エディタ（証明コンストラクタ）としてはLCF([Gordon 79]), FOL([Weyhrauch 80]), EKL([Ketonen 84]), Nuprl([Constable 86]) 等が存在する。しかし、これらの証明エディタは、ある特定の論理系に対する証明構築の支援を行うものであり、汎用性に欠ける。したがって、対象とする論理系がはっきりしている場合には適しているが上記のような一般の論理系に対する汎用の支援機能としては問題がある。

なお、我々と良く似たアプローチの仕事の例として、[Goguen 83] の institution 、[Harper 87] の Logical Framework 等の一般的な論理系の研究がある。

1.2 EUODHILOS の概要

本節では、現在開発中の論証支援システムEUODHILOSの概要を説明する。1.1節でも述べたように、論証支援システムの大きな機能として論理系を定義すること、定義された論理系に基づく証明を支援することがある。

論理系はさらに次の構成要素からなる。

(i) 言語系

論理系を定めるためにはまず論理式の構文を定める必要がある。論理式を構成する記号および構文の定義によって言語系を定める。

(ii) 導出系

論理的構造を表現するためには、公理系、導出の規則を定めなければならない。EUODHILOSにおいては導出の規則として推論規則と書き換え規則を備えている。

言語系に関しては、2節以降で詳しく説明されるため、本節では簡単に触れる程度にとどめる。本システムのユーザは構文定義支援機能を用いて問題となる対象に関する議論を行うための構文と記号を定義する。システムはこの定義に基づき論理式のパーサ、アンパーサ（プリントルーチン）を生成する。以下の処理において論理式の入力はすべてこのパーサを介してなされ、また表示はアンパーサによることになる。本節では、以下、まず導出系について説明し、その後証明支援機能について述べる。

EUODHILOSにおいて、導出を行うための出発となる公理系は公理の有限集合としてユーザが与える。すなわち、公理定義のための編集環境が提供される。この部分は単に公理を追加したり削除したりの簡単な機能である。

推論規則は自然演繹法のスタイルを採用している。すなわち、次の形式である。

$$\frac{\begin{array}{ccc}[\text{仮定式}_1] & \cdots & [\text{仮定式}_n] \\ \vdots & & \vdots \\ \text{前提式}_1 & \cdots & \text{前提式}_n\end{array}}{\text{結論式}}$$

ここで、仮定式の部分は一部または全部を省略することができる。

この形式の意味は、仮定式がついているときはその仮定式を仮定したとき、またついていないときは条件なしに、それらの結論式がすべて導かれるならば、規則の結論式が導出されるという規則を表わしている。このような形式を採用したことで自然演繹法のみならず、Hilbert 流の仮定式のない推論形態をも自然に記述可能となる。なお、推論規則には適用条件がつく場合がある。

書き換え規則は次の形式で表わされる。

$$\frac{\text{書き換え前の表現}}{\text{書き換え後の表現}}$$

書き換え規則の適用は推論規則と異なり、式全体とは限らず式の部分表現に対しても適用可能である。式のある部分表現が書き換え規則の上式と一致するとき、その部分を下式に書き換えるという形で書き換え規則は適用される。

EUODHILOS における証明は、大ざっぱにいうと、公理や仮定式から始まり、推論規則や書き換え規則を適用することで新しい結果を得ることを繰り返してなされるものである。以上の説明を図示したものが図 2 である。

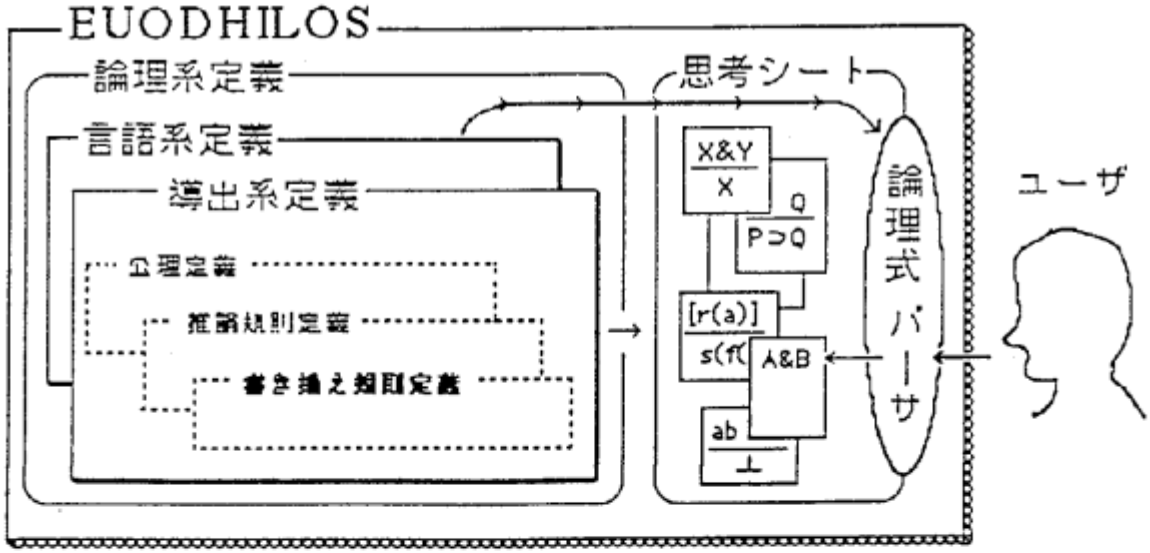

図 2 システム構成

次に、EUODHILOS における証明の様子を更に詳しく説明する。上述の様に、EUODHILOS における証明は、公理や仮定式を出発式とし、それらに推論規則や書き換え規則といった導出規則を適用して新しい定理を導出する。前にも述べたように定理およびその証明は試行錯誤的に発見されるものであるので、最初から完全な証明をユーザが与えることは期待できない。したがって、EUODHILOS における証明とは、一般には証明断片と呼ばれ

る、証明としては不完全な形式を単位とし、その合成、分解等の操作によって変形され、より完全な証明へと構成されていくものであると考えられる。

EUODHILOS において証明の構成を支援する編集環境は思考シートと呼ばれている。ユーザがこの電子式計算用紙の上で想を練り思考を行い、目的とする証明構成を行うことからこのように名付けられた。思考シートは、その上で多くの証明断片を作り、それらを組み合わせ新しい証明断片を作ったりし、また、最終的にはその上で定理を発見し、その証明を構築していくという場を提供する。このような試行錯誤的に行われる過程を支援するため、その上でなされる導出の形態もさまざまである。通常の、既に得られた式に導出規則を適用し、新しい結論式を導くという、ボトムアップ（前向き）の導出のみならず、ゴール式をサブゴール式に分割していく、いわゆるトップダウン（逆向き）の導出、また、更には、既に得られた結論部分を組み合わせ、ある証明断片の仮定となっている式を導くことで証明断片の結合を行うギャップを埋める（補間）導出形態をも支援する。現在の EUODHILOS における思考シートを用いた証明の様子を図 3 の画面例に示す。

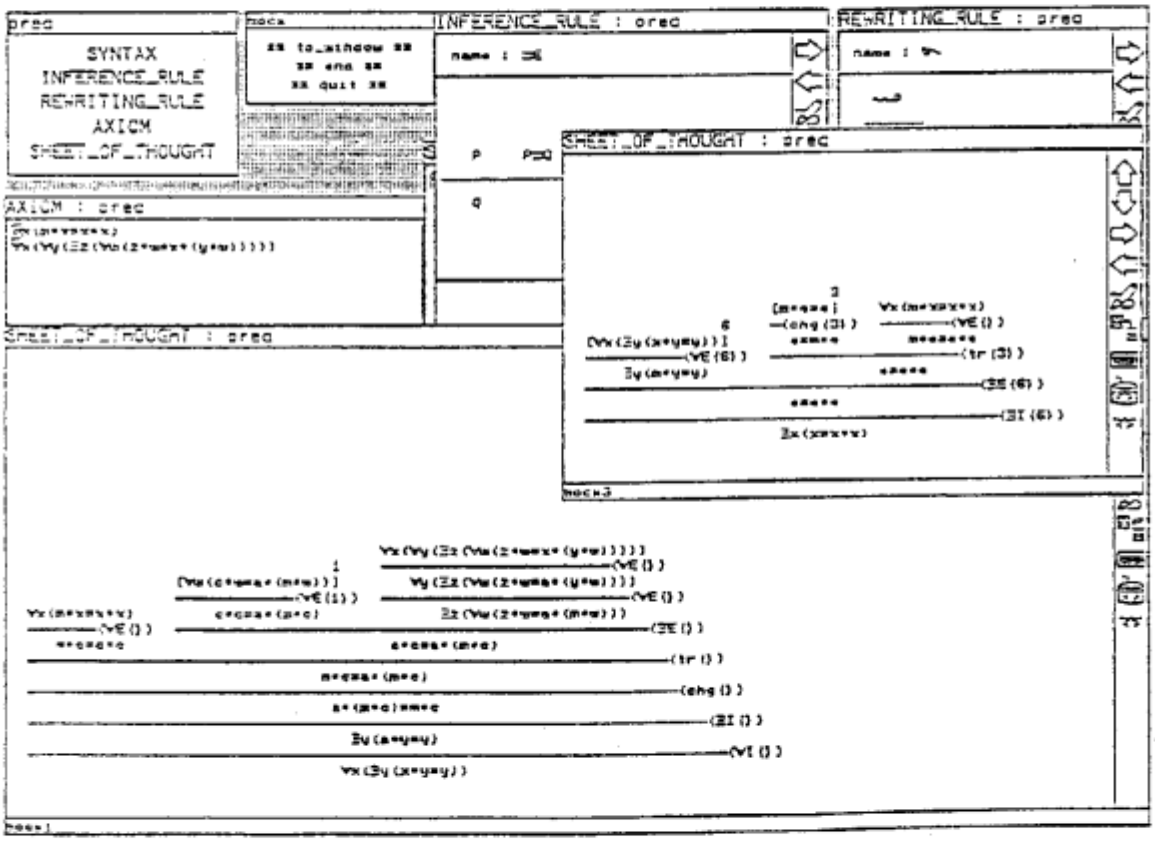

図 3 画面例

2. 論理式の構文記述

論理式の構文定義は EUODHILOS の言語系定義支援機能によって行われる。ユーザは、イメージとして持っている対象領域に関して、そこにどの様な概念が存在するのか、

また、どの様な対象があり、それらに対してどの様な操作が施されるかを考察し、それを形式的に表現することで対象をモデル化する。このような形式的表現に基づく操作を行うのが論理系であり、そこで用いられる表現形態を定めるのが言語系定義である。ユーザは自分の扱いたい対象を十分表現できるような言語系の構文規則を記述する。システムは、ユーザが記述した構文定義を元に、そのパーサ、アンパーサを生成する。その後、公理や導出規則の記述や証明の構成の際に、ユーザによって入力される論理式表現は、すべてこのパーサを用いて、その構文の正当性がチェックされ、システムの内部表現に変換されることになる。また、内部で生成された論理式はすべて、このアンパーサによってユーザに示す表現形式へと変換され表示されることになる。すなわち、パーサ、アンパーサは次のような形態でシステムとユーザの仲介に用いられるわけである。

論理式 $\xrightarrow{\text{パーサ}}$ 内部形式 $\xrightarrow{\text{操作}}$ 内部形式 $\xrightarrow{\text{アンパーサ}}$ 論理式

現在開発中の EUODHILOS において構文定義は DCG 構文([Prereira 80])を用いている。これは文脈自由文法の表現を基礎とし、非終端記号に引数を許す機能および Prolog 述語の呼び出し機能が追加されている。図 4 に一階論理の構文定義の例を示す。

論理式→論理式 1,“⇒”,論理式 1
論理式→論理式 1
論理式 1→論理式 1,“∧”,論理式 2
論理式 1→論理式 1,“∨”,論理式 2
論理式 1→論理式 2
論理式 2→“¬”,論理式 3
論理式 2→論理式 3
論理式 3→“∀”,変数,論理式
論理式 3→“∃”,変数,論理式
論理式 3→“(”,論理式,“)”
論理式 3→述語記号,“(”,項リスト,“)”
項リスト→項
項リスト→項,“,”,項リスト
項→定数
項→変数
項→関数記号,“(”,項リスト,“)”
定数→“a”～“e”
変数→“u”～“z”
関数記号→“f”～“h”
述語記号→“p”～“t”

図 4 構文記述例（一階論理）

DCG 構文による構文記述法においては大枠は文脈自由文法の形態で大きめのクラスを規定し、一般の Prolog 述語の呼び出しによって、その一部を除外することで、文脈依存部分を表わすという表現方法となる。したがって、文脈依存性をも表現できるという、一般的記述力と共に、文脈自由文法を基礎とする形態であるための表現のしやすさ、処理の容易さをも、持ち合わせている。一方 Prolog 処理系には演算子とその結合順位を定義する機能があり、それを用いた演算子順位文法の記述が可能である。演算子順位文法は、その表現が直観的であり理解が容易であるという長所があり、楽に目的の構文を記述し、また、直ちにその結果をパーサに反映させ構文の拡張を行うことができる。

論理式の構文記述の場合を考えると、その大部分(例えば、一階論理における論理演算子、∧、∨、⇒、¬等の構文) は演算子順位文法を用いて表現可能である。しかし、演算子順位文法は、論理系一般の構文記述を行うには表現力が不足している。例えば、"$\forall x\varphi(x)$"の様な表現形態は"x"と"$\varphi(x)$" の間に位置する演算子が存在しないため、演算子順位文法を用いて書き表すことができない。これら双方の表現法の短所を補い、十分な表現力と共に記述の容易さを得るためにこれらの表現を混ぜ合わせた構文記述法(以下、DCG+演算子記法と呼ぶ) を現在考慮中である。

現在考えている DCG+演算子記法においては、構文記述は構文定義と演算子定義の 2 つよりなる。演算子定義部分ではオペレータのタイプおよび結合順位を Prolog におけるオペレータ宣言と同様に与える。例えば、演算子"+"に対して"yfx,100"と与える。これを非終端記号"term"に対する演算子として用いたい場合は、構文定義の際に

```
term→term,"+",term
```

と記述する。システムは、構文定義と演算子定義の双方の情報を組み合わせてパーシング法を決定する。

DCG+演算子記法で記述された構文の定義は、演算子の結合の型と結合順位に基づいて演算子を含まない通常の DCG 形式の表現に変換された後、パーサ生成等に用いられることになる。

図 5 に図 4 の例を DCG+演算子記法を用いて表した記述例を示す。

3. パーサ生成

前述のようにユーザが定義した構文定義に基づいてパーサ、アンパーサが生成され、論理式等の表現を通じてのユーザとシステムとのやり取りは、それらを経由して行われる。現在開発中の EUODHILOS の版において、構文は DCG 構文を用い、パーサは BUP ([Matsumoto 83], [溝口 8 5])に基づいている。本節では、まず、BUP 方式によるパースの様子の説明を簡単に行った後、BUP 方式を少し変更し高速化を図る実験結果を報告する。

もともとの DCG のパーサはトップダウンパーサであるため、左再帰的な記述を含む文法の処理に問題があった。BUP はこのような問題のないボトムアップパーサである。もっとも、完全なボトムアップではなく、ボトムアップを基本に、トップダウン的な予測も加味してパーシングを進めていくという方式である。

構文定義

定数→“a”～“e”

変数→“u”～“z”

関数記号→“f”～“h”

述語記号→“p”～“t”

論理式→“∀”,変数,論理式

論理式→“∃”,変数,論理式

論理式→論理式,“∧”,論理式

論理式→論理式,“∨”,論理式

論理式→論理式,“⇒”,論理式

論理式→“¬”,論理式

論理式→述語記号,“(”,項リスト,“)”

項リスト→項

項リスト→項,“,”,項リスト

項→定数

項→変数

項→関数記号,“(”,項リスト,“)”

記号定義

“∧”: yfx, 100

“∨”: yfx, 100

“⇒”: xfx, 50

“¬”: fy, 30

図5 DCG+演算子による構文記述例

BUP 内での呼び出しは大きく分けて2種類ある。その第1のものは、文字列と、その中から切り出すべきゴール名を指定して呼び出す形式であり、第2は、すでに得られた中間ゴール名とまだ切り出していない文字列の残り部分が与えられ、それから指定されたゴール名へと切り出しを進めるための呼び出しである。前者は、述語 goal により、また後者は中間ゴール名を名前とする述語により実行される。

述語 goal は、与えられた文字列の先頭部分から切り出される中間ゴール名をまず見つけ、次にその中間ゴール名から目的のゴールへと切り出しを行う。中間ゴールから目的ゴールへと切り出しを進めるには、その中間ゴールを右辺の最左ゴールとする文法規則を見つけ、その規則の右辺の第2要素以降の切り出しを試みる。もしそれがうまくいくならば、この規則の左辺が新たな中間ゴールということになり、それから最終ゴールへの切り出しを改めて呼び出すことになる。BUP のパーシング手続きはこのように行われる。

BUPの最初の呼び出し形式は、

```
goal(ゴール,パーシングの結果,記号列,[])
```

である。第4引数はパーシング終了後の残り記号列を表わすがトップレベルの呼び出しの際は空列となる。ゴール述語は次のように定義される。

```
goal(ゴール,パース結果,記号列,最終記号列) :-
    dict(中間ゴール,中間結果,記号列,記号列残り,処理),
    link(中間ゴール,ゴール),
    call(処理),
    P=..[中間ゴール,ゴール,中間結果,パース結果,記号列残り,最終記号列],P.
```

ここで、dict は、文法の表現 $C \rightarrow C_1,\ldots,C_n$ で、C_1が終端記号であるものに対して生成される述語であり、この表現に対するパーシングが表わされている。すなわち、与えられた記号列の最初が C_1 である表現を見つけ、その処理部分を呼び出すことで、残りの文字列から $C_2,\ldots,C_n$ に対応する部分を切り出そうというものである。この、処理部分の呼び出しの前に、link 述語を呼び出すことで、現在考慮中のゴールに至るパスが存在するものであるのかをあらかじめチェックし、無駄なパーシングを行わないようにしている。最後に、得られた中間ゴールから最終ゴールへ至るパーシングを行う述語を呼び出し、目的のゴールに対応するパーシングとする。中間ゴールから最終ゴールへのパーシング述語は、文法表現 $C \rightarrow C_1,\ldots,C_n$ の C_1が非終端記号であるものに対して次のように生成される。

```
C1(ゴール,中間結果1,パース結果,記号列,最終記号列) :-
    link(C,ゴール),
    goal(C2,中間結果2,記号列,記号列2),
      ⋮
    goal(Cn,中間結果n,記号列n-1,記号列n),
    C(ゴール,中間結果,パース結果,記号列n,最終記号列).
```

この BUP パーサを高速化するために、次の2点に注目する。まず、goal述語の処理を振り返ると、その中では linkによって中間ゴールを見つけ、それを名前とする述語を生成し、呼び出す。中間述語を動的に見つける方式のため、この方式では述語生成のためのuniv述語 (=..) は必須である。動的な呼び出し述語名の生成は、処理系の最適化を難しくするものと考えられるため、まず、この部分を高速化することを考えた。それにはまず、中間ゴールから最終ゴールへとパースするための呼び出し形式を、次のように変更する。すなわち、

```
upto_最終ゴール(中間ゴール,中間結果1,パース結果,記号列, 最終記号列)
```

という形式で中間ゴールから最終ゴールへの切り出しを進めることにする。また、goal 述語の定義を各ゴール名ごとに行うことにした。その結果、各 goal 述語の定義の中のゴール名部分が定数として記述できることになった。以上の変更の結果、次のような形式のゴール述語が生成されることになる。

```
goal_ゴール(記号列，最終結果,最終文字列) :-
    goal1(記号列，中間ゴール,中間結果,残り文字列),
    upto_ゴール(中間ゴール,中間結果,残り文字列，最終結果,最終文字列).
```

ここに、goal1 の部分は、BUP における dict に相当する部分である。ここでは、与えられた記号列から始まる中間ゴールを切り出している。

次に、link 情報による枝刈の方式を変更した。BUP の方式では link 述語によって一般的なリンクの有無をチェックしている。論理系の記述に用いる点を考慮すると、それほど複雑な（深いレベルの）リンク構造は現われないものと考えられる。また、ゴールを明示した呼び出し形式に変えたため、どのゴールからどのゴールへのリンクを調べるのかがはっきりしている。したがって、ある中間ゴールからのゴールへのリンクの有無をチェックし、また、それらの中間に位置するゴールがある場合にのみリンクのチェックを行うような方式とする。また、チェックもゴールへ達する1段下の中間ゴール名だけをチェックすることとし、1段下のゴール名 subgoal ごとに次のような述語を生成する。

```
upto_ゴール(中間ゴール,中間結果,記号列，最終結果,最終記号列) :-
    \+中間ゴール==subgoal,
    upto_subgoal(中間ゴール,中間結果,記号列，途中結果,途中記号列),
    upto_ゴール(subgoal,途中結果,途中記号列，最終結果,最終記号列).
```

これらの方式の実行時間の違いをある1つの例について測定した。その例に対し、このプログラムを SUN ワークステーション上の Quintus Prolog で実行し、比較したところ、univ の変更によって 20% 程度の、更にリンクのチェック方式の変更も加えて 8 倍程度の高速化が実現できた。また推論機械 PSI 上での実験においては、インタプリタで 5 倍程度、コンパイラで 2 倍程度高速化された。PSI 上では、そのデータ構造の特徴により、univ の有無による影響はほとんどないものと推測されるため、実験結果の相違点は主にリンク構造の違いに起因するものと考えられる。原理的には、もともとの BUP 方式も我々の処理方式も同等の能力であると考えられるので、本節の結果は実験で用いられたような、文法としては比較的簡単な記述に対する高速化を実現することができたことを示しているものと考えられる。詳細な比較検討はこれからの課題である。

4. まとめ

本稿では、論理系の記述とそのパーザを中心に論証支援システムの思想、構成等について説明した。現在、論証支援システムのプロトタイプである EUODHILOS を推論機械 PSI 上で ESP 言語を用いて開発中である。なお、本稿では述べなかったが、EUODHILOS は論証のためのより良いユーザインタフェースを実現するために、フォントエディタ、ソフトキーボードによる記号定義機能、論理式の構造の把握を容易にし論理式の入力を助けるための論理式エディタ等の支援機能を持っている。論理式エディタについては文献 [沢村 8 6], [佐藤 8 6], [土屋 8 7]を、証明支援機能については [南 8 6－1], [南 8 7－1]を、また、論証支援システムの考え方全般については[南 8 6－2], [沢村 8 7], [南 8 8]等を参照いただきたい。

現在のシステムは、論証支援機能として必要最小限程度の機能しか実現していない。現在の版をたたき台とし、機能の強化、ユーザインタフェースの改良等を進めていきたいと考えている。現在のシステムには備えられていないが、今後の展開にとっては重要な課題として例えば次のような機能の取り扱いがある。

- 証明戦略
- メタ論理
- モデル

謝辞

日頃、御指導、御鞭撻を頂く国際研の北川敏男会長、並びに榎本肇所長に感謝いたします。富士通研究所の佐藤かおる、土屋恭子のお二人には、システムの実現作業の多くを頼っています、重ねて感謝します。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

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

[Constable 86] R. L. Constable et al. : Implementing Mathematics with the Nuprl Proof Development System, *Prentice–Hall*, 1986.

[de Bruijn 70] N. G. de Bruijn : The Mathematical Language AUTOMATH, its Usage, and some of its Extensions, In M. Laudet, D. Lacombe, L. Nolin and M. Schutzenberger (eds), *Symposium on Automated Demonstration*, Springer–Verlag, December 1970.

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

[Goguen 83] J. A. Goguen and R. M. Burstall : Introducing Institutions, *LNCS* 164, Springer–Verlag, 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 and G. Plotkin : A Framework for Defining Logics, *ECS-LFCS-87-23*, University of Edinburgh, 1987.

[Ketonen 84] J. Ketonen and J. S. Weening : EKL — An Interactive Proof Checker, User's Reference Manual, *Dep. of Computer Science*, Stanford University, 1984.

[Lakatos 76] I. Lakatos : Proofs 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] S. K. Langer : A Set of Postulates for the Logical Structure of Music, *Monist* **39**, 1925.

[Matsumoto 83] Y. Matsumoto, H. Tanaka, H. Hirakawa, H. Miyoshi and H. Yasukawa : BUP : A Bottom-Up Parser Embedded in Prolog, *New Generation Computing* **1**, 1983.

[Pereira 80] F. C. N. Pereira and D. H. D. Warren : Definite Clause Grammars for Language Analysis — A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artificial Intelligence* **13**, 1980.

[Weyhrauch 80] R. W. Weyhrauch : Prolegomena to a Theory of Mechanized Formal Reasoning, *AI Journal* **13**, 1980.

[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.

[溝口８５] 溝口他：Prolog とその応用２、総研出版、１９８５年。

[沢村 ８６] 沢村、南他：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。

[南８８] 南、沢村他：論証支援システム：論理モデル構築のための支援ツール、LPC'88、ICOT, １９８８年４月。