Lukasiewiczの含意を用いたファジィ論理プログラミング

メタデータ	言語: jpn 出版者: 公開日: 2012-05-24 キーワード (Ja): キーワード (En): 作成者: 安井, 浩之 メールアドレス: 所属:
URL	http://hdl.handle.net/10291/12898

明治大学大学院　理工学研究科

1995年度

博士学位請求論文

Lukasiewiczの含意を用いた ファジィ論理プログラミング

指導教員　向殿 政男 教授

学位請求者　基礎理工学専攻 情報科学系

安井 浩之

もくじ

1 序論 **4**

1.1 本研究の目的 4

1.2 これまでの研究の流れ 7

2 ファジィ論理プログラミングの定義 **10**

2.1 論理プログラミング 10

2.1.1 論理プログラミングの定義 11

2.2 ファジィ論理プログラミング 14

2.3 プログラムに与えられる真理値 16

3 ファジィ論理プログラミングにおける推論規則について **20**

3.1 推論規則の決定 20

3.2 推論規則における含意演算と連言演算の満たすべき性質 - 要請 1 21

3.3 推論規則における含意演算と連言演算の満たすべき性質 - 要請 2 24

3.4 ファジィ論理プログラミングに適した含意演算と連言演算 27

4 推論規則の完全性と健全性 **30**

4.1 導出規則 30

4.2 導出規則の完全性と健全性 32

5 言語ヘッヂを導入したファジィ論理プログラミングについて **36**

5.1 言語ヘッヂの定義 36

5.2 ヘッヂを持つファジィ論理 39

5.3 言語ヘッヂを持つファジィ論理プログラミングの推論規則 41

5.3.1 指数型ヘッヂ . . . 41
5.3.2 単調ヘッヂ . . . 45
5.3.3 その他のヘッヂ . . . 47
5.4 拡張導出規則の完全性と健全性 . . . 50

6 Łukasiewicz の含意に基づくファジィ Prolog(LbFP) 52

6.1 ユーザの要請分析 . . . 52
6.2 Prolog システム . . . 55
6.3 LbFP システム I - 純粋ファジィ論理プログラミング部 - . . . 57
6.3.1 再帰的規則に対する制限 . . . 57
6.3.2 真理値 . . . 58
6.3.3 言語ヘッヂ . . . 59
6.4 LbFP システム II - 非ファジィ論理プログラミング部 - . . . 60
6.4.1 正規化 . . . 61
6.4.2 レベル n ファジィ集合 . . . 61
6.4.3 ファジィ項 . . . 62
6.4.4 ファジィ集合の基数 . . . 62
6.4.5 ファジィ量限定 . . . 63
6.4.6 その他の LbFP 固有の機能 . . . 64

7 LbFP を用いたファジィ問題の解決 66

7.1 ファジィ推論 . . . 66
7.1.1 簡単なファジィ推論 . . . 66
7.1.2 岩石分類システム . . . 68
7.2 宣言的ファジィアルゴリズム . . . 71
7.2.1 簡単な宣言的ファジィアルゴリズム . . . 71
7.2.2 ファジィオートマトン . . . 72
7.2.3 病気の状態と薬の投与に関するファジィオートマトン . . . 73

8 結論 75

A 証明 81

B LbFP 文法 **84**

C LbFP プログラムと動作例 **88**

C.1 岩石分類システム 88

C.1.1 ソースコード 88

C.1.2 動作例 90

C.2 ファジィオートマトン 91

C.2.1 ソースコード 91

C.2.2 動作例 92

D LbFP システム ソースコード **95**

D.1 メインと構文解析部 95

D.1.1 fpro.yacc 95

D.2 字句解析部 133

D.2.1 fpro.lex 133

D.3 ヘッダファイル 134

D.3.1 fpro.h 134

D.3.2 version.h 137

第 1 章

序論

1.1 本研究の目的

人類史上最も有用な道具であるといえるコンピュータは，人間を単調な仕事から解放し，大量のデータ処理を扱い，高速で計算を行うという能力を持っている．今日，コンピュータの果たす役割はかつて存在した如何なる工学的発明をも凌いでおり，理工学的分野にとどまらず，文学，法学，経済，農業，さらには日常生活の隅々にまで広く深く浸透している．

コンピュータは「電脳」と呼ばれるとおり，一般的には，コンピュータそのものを電子的な人工頭脳と見なす場合が多い．しかし，本当はコンピュータは電脳ではなく電子計算機に過ぎない．これほど高速な高機能なコンピュータでも，数値計算の速度を除いては，人間の頭脳には遥かに及ばないのである．この理由は簡単で，人間の頭脳のシステム，特に知能に関して解明されていないためである．コンピュータの仕事は，対象問題を手順に従って解くことであり，対象問題を解く方法が分からないような場合は無力である．

人間の知能（論理）を形式化するための研究は，コンピュータに知能を持たせようとする企てよりも遥か以前，古代ギリシア時代から論理学者たちによって行われていた．これらの研究の成果が２値論理であり，物事は真か偽に決定できるという場合における人間の論理（厳密論理）の形式化がほぼ成功した．

コンピュータに知能を持たせようとする人工知能の研究は，人間の知識を２値論理の論理式で表現出来るという仮定の基で当初，２値論理，及び２値論理に基づく推論を基礎に始められた．中でも J.A.Robinson により提案された導出原理 [22] はこの分野で最も重要な研究である．この導出原理を用いることにより，コンピュータを利用して推論や定理の証明を行うことが可能となっ

た．人工知能用のコンピュータ言語として，もっとも広く用いられているは Prolog はこの導出原理を基に推論を行っている．

Prolog は 1 階述語論理に基づく論理型プログラミング言語であり，その数学的基礎理論は論理プログラミングと呼ばれる．論理プログラミングは論理式をプログラミング言語として見なし，コンピュータで自動的に進行できる推論規則を用いることで推論を行い，処理を実行する．つまり，論理プログラミングは知識の中にある対象間の関連から推論によって答えを導き出すという人間の知能をコンピュータで実現するための理論である．この論理プログラミングの登場によって，人工知能の研究が大きく進んだといえる．

しかし，人間の論理には，真か偽のどちらかに必ず決定されない曖昧な場合も存在しているし，現実的にはこのような場合のほうが多いといえる．近代になって，このような人間の論理（曖昧論理）を形式化するという目的のために，直観主義論理や多値論理などの研究がなされてきたが，最も大きな業績を上げたのが L.A.Zadeh により提唱されたファジィ集合論である．[31]

ファジィ集合論に基づくファジィ理論では，真か偽かがはっきり出来ないような人間の曖昧な知識を, 曖昧さの度合を真を 1，偽を 0 として，閉区間 $[0,1]$ 上の実数値 (真理値) で表現し，２値論理では対応仕切れなかった多くの曖昧な問題を形式化することに成功した．

当然，人間の曖昧論理を扱うために，論理プログラミングの理論にもファジィ理論を導入することが考えられたが，ファジィ集合論により規定されるファジィ論理では，２値論理で成立した導出原理が成り立たないという大きな問題が存在していた．

この試みに始めに成功したのは R.C.T.Lee[16] である． R.C.T.Lee は知識に割り当てられる真理値を 0．5 以上に制限することで 2 値論理の導出原理をそのままファジィ論理に導入した．しかし，これでは閉区間 $[0,1]$ 上の任意の実数値を真理値とするファジィ論理の特徴が犠牲にしてしまうことになる．

向殿ら [19] は, この制約を無くすため, 推論規則そのものをファジィ論理向けに拡張し, 任意の真理値を割り当てることの出来るファジィ導出原理を提案した．ここでは, 2 値論理の導出原理では $L_1 \vee a$ と $L_2 \vee \neg a$ から $L_1 \vee L_2$ を得たが, 拡張された導出原理ではファジィ導出節 $L_1 \vee L_2 \vee (a \wedge \neg a)$ を得る．項 $(a \wedge \neg a)$ は, この推論自体の矛盾の度合をあらわしている項と解釈しており, 真理値を $\{0,1\}$ に限れば必ず 0 になるので，この拡張された導出原理は 2 値論理の導出原理を含んでいる. しかし, 推論のキーになる項 (この場合は a) の真理値が決定されないと推論結果の真理値を求めることが出来ない．このことは, 論理式の他に, キーとなる項にも真理値を与えなければならないことを意味する. ファジィ導出原理では この真理値を与えるために，ファジィ論理では

説明できないような手続きを採用しているが，やはり，ファジィ論理から逸脱していると言わざるを得ない．

これらの推論規則の問題点は，論理式を2値論理と同じ節形式としている点であると考えられる．

もともと，推論は含意演算（「…ならば〜である」という意味に解釈される論理演算）を用いて行なわれていたものである．節形式での論理式表現は，2値論理における論理式の標準形に対応しており，含意演算も節形式で表すことが出来たため，2値論理における推論規則の一般形として節形式の論理式の推論規則である導出原理が用いられていた．ところが，ファジィ論理における含意演算は1つに確定されておらず，2値論理のように節形式で表現できないものがほとんどである．このことからも，ファジィ論理プログラミングにおける推論を考える上で，2値論理と同じ節形式を用いる必然性はないということできる．また，ファジィ論理の多様な含意演算を用いることで，推論の意味づけも多様になり，応用範囲が広がることも予想される．

本論文では，1階ファジィ述語論理に基づいて，含意演算を用いた論理式表現におけるファジィ論理プログラミングについて定義，考察し，そのファジィ論理プログラミングに基づくファジィPrologを提案することを目的としている．さらに，提案するファジィPrologをより実用性の高いものとするために，ユーザのファジィPrologに対する要請を分析し，ファジィ量限定子のような1階ファジィ述語論理の範囲では説明できないが必要とされるような機能の導入を行い，比較的簡単にファジィ問題を解決できるものとして行くことを目的にしている．

本論文の構成は，以下のようになっている．

まず次の節で，これまでのファジィ論理プログラミングの研究の流れを示す．

2章では，本論文で考察するファジィ論理プログラミングの諸定義を行う．

3章では，ファジィ論理プログラミングにおける推論規則に関する考察を行い，推論において最も重要な役割を果たす含意演算について，いくつかの側面から検討し，本論文題目中にあるŁukasiewの含意がファジィ論理プログラミングに最もふさわしい含意演算であることを示す．

4章では，提案するファジィ論理プログラミングにおける推論動作の保証を与えるために，推論規則の完全性と健全性を証明する．

5章では，ファジィ論理プログラミングにおける論理式の表現力をより豊かにするために，1階ファジィ述語論理に言語ヘッヂと呼ばれる1項論理演算群を導入し，ファジィ論理プログラミングを拡張する．

6章では，本研究に基づくファジィPrologであるŁukasiewicz's implication based Fuzzy Pro-

log (LbFP) を提案し，その言語仕様や動作仕様を示す．さらに，ファジィ量限定子のような１階ファジィ述語論理の範囲では扱えないような機能を導入し，ファジィ理論における諸問題にも対応できるような言語仕様を導入する．

７章では，LbFP の応用例として，ファジィ推論やファジィアルゴリズムのプログラム記述ならびに動作を示す．

８章で，本論文の総括として，まとめと今後の課題を示す．

1.2 これまでの研究の流れ

本論に入る前に，ファジィ論理プログラミングに関する過去の研究の流れを示しておく．

ファジィ論理プログラミングの研究には, 大きく Lee の自動推論からの流れとそれ以外に分けることが出来る．

ファジィ論理プログラミングの研究の先駆けである R.C.T.Lee [16] はファジィ論理における自動推論のために, J.A.Robinson によって提案され，２値論理における推論規則の一般形として認められていた導出原理 [22] をファジィ論理に導入することを試み, 真理値が 0．5 以上の論理式であれば, 導出原理が成り立つことを示した．

導出原理とは，知識を表現する論理式に節形式表現を用いた場合に適用出来る推論規則である．節形式とは，命題論理の範囲内でいうならば，$L_1 \vee L_2 \vee \cdots \vee L_n$ のように論理式中の全てのリテラル (正リテラルでも負リテラルでもよい) が，選言結合されているような（和形式）論理式をいう．各知識 C_i が $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ のように連言結合されていると見なされるので，全ての知識をまとめて考えると，和積形式の１つの論理式と見なすことが出来る．

任意の２値論理関数は和積形式に変換可能であるので，節形式で適用できる推論規則は，どんな２値論理関数で表現された知識に対しても適用可能ということになる．これが，２値論理における推論規則の一般形として認められる理由である．

R.C.T.Lee の研究では, 導出原理を適用した結果の真理値 T_d は, 推論の前提となる式を C と C' とすると, $0.5 \leq \min\{T(C), T(C')\}$ ならば, 常に $0.5 \leq \max\{T(C), T(C')\} \leq T_d \leq \max\{T(C), T(C')\}$ となり, この推論法が健全性を満足することを示している．健全性とは，この推論規則を適用して得られた推論結果が他の知識と矛盾しないことを保証する性質である．

R.C.T.Lee の研究を初めて論理プログラミングシステムに応用したものが, 石塚らによる Prolog-Elf[6] である．Prolog-Elf ではその実装において，ユーザ定義の閾値を設定することで無駄な推論

を省き, 効率の高い推論を実現している. この閾値の概念は後のファジィ論理プログラミングシステムの実装に大きな影響を与えた.

R.C.T.Lee の真理値 0.5 以上というの制限を解消したのが, 向殿らのファジィ導出原理 [19] である. ファジィ導出原理では, 2 つの節形式の式 $L_1 \vee a$ と $L_2 \vee \neg a$ からファジィ導出節 $L_1 \vee L_2 \vee (a \wedge \neg a)$ を得るという推論規則である. $(a \wedge \neg a)$ なる積項は相補項と呼ばれ, その真理値はこの推論の信頼度を意味すると考えた. しかしファジィ導出節 (推論結果) の真理値を知るためには, 相補項の真理値を知る必要がある. ところが, この推論規則では相補項の取る真理値を知ることが出来ないという問題点があった.

ファジィ導出原理を導入した Z.L.Shen らのファジィ Prolog[24] では, 相補項 $(a \wedge \neg a)$ の真理値を求めるために, 確信度に相当する $[-1, 1]$ の実数値 w を論理式に与えている. 規則式の前件部の真理値 (各リテラルの真理値の最小値)T_b と後件部の真理値 T_a に対して, $w = 2(T_a - 1) \times 2(T_b - 1)$ という関係式を与えることとする. すると, 推論過程で前件部の真理値 T_b は決定されるので, この関係式より, 後件部の真理値 T_a を求めることが出来る. しかし, w の値とファジィ論理における真理値や確信度との関係が明らかにされていないと言う問題点が指摘されている.

これらのファジィ論理プログラミングシステムに先駆けて提案されたのは, R.C.T.Lee の流れとは別の R.A.LeFaivre の FUZZY[17, 2] であった. FUZZY は 2 値の論理プログラミングの各論理式に $[0, 1]$ の値を割り当てたものであり, ファジィ論理プログラミング言語というよりは, 重み付き 2 値論理プログラミング言語であった. なお, 重み付き 2 値論理プログラミングとファジィ論理プログラミングとは次のように区別される.

- 重み付き 2 値論理プログラミング
  - 論理式全体に真理値が与えられた時, 式中の個々のリテラルが取る真理値は考慮しない.
    例えば, $A \wedge B$ の真理値 $T(A \wedge B)$ は $T(A \wedge B) = T(A) \wedge T(B)$ である必要はない.
- ファジィ論理プログラミング
  - 論理式全体に真理値が与えられた時, 式中の個々のリテラルが取る真理値も考慮しなければならない. すなわち, 式の真理値は各リテラルの真理値から計算できる (真理関数的性質を持つ) としており, 例えば, $A \wedge B$ の真理値 $T(A \wedge B)$ は $T(A \wedge B) = T(A) \wedge T(B)$ でなければならない.

M. Van Emden[4] は前件部の各真理値と規則式に与えられた真理値との代数積を取って, 推論結果の真理値としている重み付き 2 値論理プログラミングである. Lee のような, 真理値が 0. 5 以上でなければならないという制約はない.

T.P. Martin らの FPROLOG[18] は, 規則式には真理値を与えず, 事実式のみに真理値を与えるというファジィ論理プログラミングを提案している. M. Van Emden の規則式に与える真理値を 1 とした場合とほぼ等しいものであるが, 推論結果の真理値の計算方法は代数積に限らず, ユーザ定義の連言演算 (通常は $\min\{\cdots\}$) を用いることが可能であり, その点が違いであるといえる. いずれにしても, これらの論理プログラミングにおいて, 含意 ($\leftarrow$) は論理演算というより, 形式文法における書き換えを表す記号というイメージに近い.

菊池らの PROFIL[8] は, 節形式の論理式の問題点に着目し, 含意演算に Gödel の含意を用いた論理式を用いており, 真理値としてその論理式がとる下限値を採用した. また, その推論規則として 2 値論理で用いられるような線形導出を示し, 線形導出が完全性かつ健全性を満足することを示した. [9]

線形導出とは, 節形式論理式に含まれる正リテラルが高々 1 つしかないように制限することで, 導出原理を効率よく適用し, 推論結果を得るための手法である.

その他, ファジィ集合を述語の項として扱う馬野の FS-Prolog [26] や, 様相論理に基づく C.J. Hindel の Fuzzy Prolog [5] のような言語真理値を扱うファジィ論理プログラミングシステムなどがある.

また, 本論で提案しているファジィ論理プログラミングと同様に, Łukasiewicz の含意を用いているものとして, Łukasiewicz 論理の演算を応用した J. Ivánek らの Prolog に似たエキスパートシステム [7] やŁukasiewicz 論理における多値論理プログラミングである F. Klawonn らの LULOG [12] がある.

J. Ivánek らのシステムは, 元々エキスパートシステムとして考えられたもので, 規則に真理値とは別に重みという度合が与えられており, 本論におけるの解釈とは異なったものとなっている. また, 異なる規則から同じ推論結果が得られた場合の真理値の計算にŁukasiewicz の和演算 ($\min\{a+b,1\}$) を用いているが, 本論文におけるファジィ論理プログラミングはファジィ理論に基づいて, 論理和 $\max\{a,b\}$ を用いており, この点が異なっている.

また, F. Klawonn らの LULOG は論理演算の積演算 $a \wedge b$, 和演算 $a \vee b$ がそれぞれŁukasiewicz の積演算 $\max\{a+b-1,0\}$, 和演算 $\min\{a+b,1\}$ となっており, 真理値の解釈も有限の多値を前提としたものとなっているという点で, ファジィ論理プログラミングとはいえないものである.

第 2 章

ファジィ論理プログラミングの定義

本章では，本論文で取り扱うファジィ論理プログラミングの枠組みを定義する．

2.1　論理プログラミング

アルゴリズムという語の厳密な定義は「与えられたある型に属する問題を解く手段とか手続きのことで，その手続きは有限回で終了せねばならない」というものである．問題によっては有限回で終了しないこともあるが，その場合はその問題を解くアルゴリズムが存在しないことを意味する．

一般的にはアルゴリズムというと，問題を解くときの計算処理とその流れを指すものであり，どのような計算をどのような手順で行っていくかを細かく規定しなければならない．

これに対して，R. Kowalski は「アルゴリズム＝論理＋制御」[14] という考えを提唱した．これは，アルゴリズムは問題を捕える論理と，論理をどう展開して答えを得るかという制御に分けられるというものである．

論理プログラミングとは「問題を解くアルゴリズムを得る上で人間が果たす中心的役割は論理を決定することであり，その論理をどのように用いて処理を進めていくかの制御は決定することはさほど創造的でないのでコンピュータでも充分行うことが出来る」というような考えから発想されたプログラミングの自動化を目指した理論である．

2.1.1 論理プログラミングの定義

論理プログラミングで扱う論理は1階述語論理である．1階述語論理の細かい定義は[20]を参照していただきたい．ここでは，論理プログラミングに関係する部分だけを紹介する．

定義 1 解釈
論理式に解釈を与えるとは，論理式中に現われる全てのリテラルに対して，真か偽を与えることをいう．

論理プログラミングにおいて，対象となる問題の論理は論理式で表現される．これを公理と呼ぶ．

定義 2 公理
公理とは常に真でなければならない論理式である．

一般的に論理式は，解釈の与え方によって真になったり偽になったりする．実はこの公理の定義は解釈に制限を与えるものであり，公理となっている論理式が偽になるような解釈は許されないということを意味している．

定義 3 妥当
公理を真にする解釈を妥当な解釈と呼び，妥当な解釈において真となるような論理式のことを妥当な論理式 *(妥当式)* と呼ぶ．

妥当な論理式とは，この公理において常に正しい論理式であり，その論理式が表現することが正しいということを意味する．

対象となる問題の解答は，その問題において正しいであるはずなので，解答を表わす論理式は妥当な論理式となっている．

次に制御の方を考える．制御が行うのは対象となる問題の論理をどのように用いて処理を進めていくかである．この処理を進めるために必要な手続きが推論規則である．

定義 4 推論規則
推論規則とは，1つ以上の論理式から1つの新たな論理式を生成する手続きをいう．

そしてこの推論規則を使って公理から得られる新しい論理式を定理と呼ぶ．

定義 5 定理

- 公理に推論規則を適用して得られた論理式.

- 公理と定理に推論規則を適用して得られた論理式.

この定義によるものだけを定理と呼ぶ.

一般的に良く知られている推論規則に modus ponens というものがある. これは, 論理式 A,$A \to B$ から B を得るものである. また, 論理式中に含まれる変項に別のものを代入する操作も推論規則の一種である.

定義 6 代入
論理式 χ 中の同一変項を全て別の同一定項又は変項に置き換える操作 θ のことを**代入**と呼ぶ. また, 代入操作後の論理式は $\chi\theta$ と表される.

これら 2 つを同時に使う有名な例を示す.

例 1 いま公理として

	対象問題の論理	*1* 階述語論理式
A_1	「ソクラテスは人間である」	$human(Socrates)$
A_2	「全ての人間は死ぬ」	$\forall(x)(human(x) \to die(x))$

が与えられているとする. このとき, この問題において「ソクラテスは死ぬ」 $die(Socrates)$ という定理が次のようにして得られる.

1. 公理 $A_2 = \forall(x)(human(x) \to die(x))$ に x に $Socrates$ を代入する操作 θ を行うと, 定理 $T_1 = A_1\theta = human(Socrates) \to die(Socrates)$ が得られる.

2. 公理 $A_1 = human(Socrates)$ と定理 $T_1 = human(Socrates) \to die(Socrates)$ に *modus ponens* を用いると, 定理 $T_2 = die(Socrates)$ か得られる.

もし推論規則がいいかげんなものだとすると, その結果得られる定理も正しいかどうか分からない. 推論規則が正しいものであることを示すには, 得られる定理が正しいかどうかを示せばよい.

完全性 論理式が公理において妥当であれば, その公理から推論規則で導き出される定理である.

健全性 論理式が, 公理から推論規則で導き出された定理であれば, その公理において妥当である.

この2つの性質をもった推論規則であれば，定理は正しく，しかも公理において妥当な論理式はすべて定理であるということがいえ，対象となる問題の解答が定理となっていることが保証されるわけである．これにより対象となる問題の解答が，推論規則を用いることで得られるということが言えるのである．

J.A.Robinson によって示された導出原理 [22] はこの完全性と健全性を満足するような推論規則である．

導出原理で扱う論理式は節と呼ばれ，$L_1 \vee L_2 \vee \cdots \vee L_n$ というような節形式という形式になっている．L_i はリテラルまたは $\neg$ のついたリテラルを意味する．

1階述語論理において，任意の論理式はこの和積形式に書き換えることが出来ることが知られている．積和形式とは, いくつかの節形式論理式を $\wedge$ で結んだ形をした式のことである．

論理プログラミングにおいて，各公理 A_j は全て連言結合していると考えられるので，対象となる問題の論理全体 P は $P = A_1 \wedge A_2 \wedge \cdots \wedge A_m$ となる．この P を和積形式に書き換えた結果を $P = A'_1 \wedge A'_2 \wedge \cdots \wedge A'_l$ とすると，A'_k は全て節形式になるので, 導出原理の適用が可能になる．

つまり，任意の問題は導出原理で扱うことが出来るということになる．

導出原理

C_1，C_2 を節と，L_1，L_2 をリテラルとする．このとき $L_1\theta_1 = L_2\theta_2$ なる代入 θ_1，θ_2 が存在しているとき，2つの節 $C_1 \vee L_1$，$C_2 \vee \neg L_2$ から節 $C_1\theta_1 \vee C_2\theta_2$ を得る．

この導出原理を用いて，定理として目的である対象問題の解答を表わす節を得ることが出来れば問題が解けたことになる．しかし導出原理を闇雲に用いても求める節にたどり着く保証はないし，求める解答が間違っているとなると永久に解けないことになる．

反駁法は効率よく定理が導けるかどうかを，知ることが出来る手法である．反駁法は一般に背理法と呼ばれ，定理として求めたい式の否定を公理に加えたときに，公理から空節が導かれた場合に求めたい式が定理であることがいえる，というものである．

導出原理の規則で行くと節 L と $\neg L$ から得られる節は何もないことになる．この何もない節を空節と呼ぶ．空節は偽を意味する．しかし反駁法にしても，有限回で空節が得られるという保証はない．しかし，もし公理がすべて Horn 節であるときは，求めたい論理式が定理であるかどうかを，有限回の導出原理の適用で知ることが出来ることが，証明されている．Horn 節とは，節に含まれる否定のついていないリテラルの数が，1つだけしかないもの呼ぶ．つまり論理式が Horn

節である場合は，論理プログラミングはアルゴリズムといえる訳である．

Horn 節は $L \lor \neg L_1 \lor \cdots \lor \neg L_n$ という形をしているが，これは $L_1 \land \cdots \land L_n \to L$ を意味しており「もし L_1 かつ $\cdots$ かつ L_n ならば L である」というような IF-THEN ルールを表わす論理と考えることが出来る．

2.2 ファジィ論理プログラミング

一般に論理プログラミングで扱われる論理式は Horn 節形式で，$a \leftarrow b_1, \cdots, b_m$ という形と，含意演算のない a のような形をしている．前者は $a \leftarrow b_1 \land \cdots \land b_m$ を意味し，**規則式**と呼び，後者を**事実式**と呼ぶ．また，含意演算の左側の項 (a) を**後件部**または**頭部**と，右側の項 $(b_1 \land \cdots \land b_m)$ を**前件部**または**本体**と呼ぶ．含意演算が一般的な表記と逆向きとなっているが，これは論理プログラミングの考案者である R. Kowalski の表記法 [13] に由来したものである．

本論で扱うファジィ論理は一階ファジィ述語論理であり，一階述語論理を真理値に関してファジィ拡張したものである．よって，一階ファジィ述語論理における以下の諸定義は，真理値に関するものを除いて 2 値の一階述語論理における諸定義に準ずるものである．[20]

定義 7 一階ファジィ述語

一階ファジィ述語とは，*2* 値論理における一階の述語の取る真理値を $\{0, 1\}$ から $[0, 1]$ へ拡張したものである．

なお，述語の変数を束縛する束縛子は全称束縛子 $\forall$，存在束縛子 $\exists$ のみとし，most などのファジィ量限定子は 1 階述語の範囲では扱うことができないので，考慮しないものとする．

本論のファジィ論理プログラミングで扱う論理式の形式は前述の通りで，このような論理式を以下のようにプログラム式と呼ぶこととする．

定義 8 プログラム式

- $Q\phi \leftarrow \psi_1 \land \cdots \land \psi_m$
- $Q\psi$

ただし，ϕ, ψ_i は一階ファジィ述語を，Q は束縛子 *(*$\forall$ もしくは $\exists$*)* を表すものとする．

特に断らない限り，以後，式とはこのプログラム式を意味するものとする．

定義 9 プログラム式は閉論理式とする．つまり，ファジィ述語中の変項はすべて全称束縛子か存在束縛子によって束縛されている．さらに，存在束縛子は *Skolem* 化によって排除することとする．

定義 10 全称束縛子で束縛されたプログラム式の真理値はその下限値で定義される．

$$T(\forall(x)\ \phi(x) \leftarrow \psi(x)) = \inf_{\forall a}\{T(\phi(a) \leftarrow \psi(a))\} \tag{2.1}$$

この場合，下限値はエルブラン集合における全ての解釈において求める．つまり，式中の x に代入できる全ての定項に関して式の取る真理値を求め，その下限値で真理値を定めるということである．

エルブラン集合は，一般に可算無限なの inf となっている．

プログラム式は閉論理式であり，存在束縛子は skolem 化によって全称束縛子で束縛されている変数に関する関数に書き換えられているので，式中に現れる変数は必ず全称束縛子で束縛されている．そこで以後，式中の変数が明らかである場合は表記を簡単にするために全称束縛子を記述しないこととする．

2 値の論理プログラミングでは，知識として与えられた論理式（公理）は，真 (1) の真理値を持っていた．しかし，ファジィ論理においては，真理値は閉区間 $[0,1]$ の任意の実数値が認められているので，公理は，個別の真理値を持つことになる．ただし，2 値論理プログラミングでは公理に含まれない知識（論理式）は偽 (0) と考えられるので，ファジィ論理プログラミングにおいても同様に考え，真理値 0 は公理に与えられないものとする．

定義 11 公理

ファジィ論理における公理とは，$(0,1]$ 上の任意の実数値を割り当てたプログラム式である．また，公理の有限個の集合を公理系と呼ぶ．

ここで，ファジィ論理プログラミングにおける推論規則について述べておく．ファジィ論理プログラミングは 2 値論理プログラミングと異なり，各定理が各々真理値を持っている必要がある．そのため推論規則において，推論結果の真理値を決定する必要がある．

定義 12 推論規則

推論規則とは，1 つ以上の式 *(推論の前提)* と各式に与えられた真理値から，1 つの新たな式とその真理値 *(推論の結果)* を生成する手続きをいう．

この推論規則の定義より，ファジィ論理プログラミングにおいては，定理も公理と同様に真理値が与えられたものになる．

定義 13 プログラム
真理値を与えられたプログラム式の集合をプログラムと呼ぶ．また，プログラム Γ が与えられ得た時，プログラムの要素の式 χ に与えられた真理値を $\mu_\Gamma(\chi)$ で表す．

つまり，公理系もプログラムである．また，推論規則によって得られた新たな式も，真理値を与えられた式であるのでプログラムと見なされる．

定義 14 サブプログラム
プログラム Γ において式 ψ を推論規則を複数回プログラムに適用して得られた推論結果とする．このとき，その推論課程に関わる全てのプログラム式の集合を ψ に関するサブプログラムと呼び，Γ_ψ で表わす．

このとき，プログラム Γ から ψ に関するサブプログラム Γ_ψ を削除すると ψ は Γ から推論出来なくなる．

2.3 プログラムに与えられる真理値

ファジィ論理プログラミングにおける推論規則を考察するためには，形式的（手続き的）に求められた定理が，意味的に正しいかを考慮する必要がある．意味的に正しいとは，公理の式中の各リテラルに具体的に解釈を与えたとき，その公理の真理値に矛盾しないかということである．意味的に正しいことを 2 値論理の場合と同様に妥当と呼ぶこととする．

つまりこの場合，推論規則によって得られた推論結果の真理値に基づいて解釈を行った場合に，公理系全体に対して妥当な解釈となるかどうかを考慮するということである．この問題に関する考察は，次章で行う．

ここで妥当を定義するために，本論においてプログラムに与えられる真理値の持つ意味について明確にする必要がある．具体的言えば，知識に割り当てられた真理値に矛盾する解釈とはどのような場合であるかを明らかにするということである．

プログラムに与えられる真理値の持つ意味は大きく分けて次の 3 つに分類される．

下限値： 真理値は，解釈を与えたときプログラム式が取る下限の値を規定するものである．

上限値： 真理値は，解釈を与えたときプログラム式が取る上限の値を規定するものである．

確定値： 真理値は，解釈を与えたときプログラム式が取る唯一の値を規定するものである．

ここで注意する点は，解釈によって論理式中の各リテラルに与えられる真理値は唯一の値であるということである．

では，ファジィ論理プログラミングではどれを適用すべきであろうか．

確定値は，その値以外は認めないものである．例えば，「x は若い」という述語を考えた場合，T(tom は若い) = 0.8 という場合，tom の若い度合は 0.8 であり，他に tom の若い度合は 0.85 であるという情報があれば，それは矛盾しているということになる．

論理数学的にはこの確定値がもっとも良いといえるが，人間の感覚から言えば厳密すぎる．その上，1 つの情報に対していくつかの情報源が存在すると，すぐに矛盾をきたすことが予想される．この点から考慮して真理値は確定値ではないものとする．

下限値は，与えられた真理値よりもより真に近い解釈を許すものなので，「少なくともこれ以上は正しいことを保証する」と解釈でき，推論を行うことは，知識の正しさを追及していくことに相当する．一方，上限値は，「多くともこれ以上正しいことはないことを保証する」と解釈できるため，推論を行うことは，知識の正しくない度合を追及していくことに相当する．各々プラス思考の推論，マイナス思考の推論ということができる．

これらの内どちらがファジィ論理プログラミングに適しているかを考察する．

ファジィ論理プログラミングは真理値が 0 と 1 に限られた場合，2 値論理プログラミングと考えられるので，2 値論理プログラミングの場合は真理値の意味はどちらの場合であるかを考える．

2 値論理プログラミングにおいて，「知識が存在する＝知識は真である」という立場は取っているが，「知識が存在しない＝知識は偽である」という立場は取っておらず，「知識が存在しない＝知識は真か偽か不明である」という立場を取っている．このことから 2 値論理プログラミングにおいては，真理値は上限値ではなく下限値であるということが出来るので，必然的にファジィ論理プログラミングの真理値も下限値であるということできる．

よって，妥当に関して次のように定義することが出来る．

定義 15 妥当な解釈

妥当な解釈 I とは，真理値を割り当てられた式に解釈 I を与えた時に次の式を満足するものを言

う．ただし，$/\chi/_I$ は式 χ に解釈 I を与えた時の真理値を，$\mu(\chi)$ は式 χ に割り当てられた真理値を意味する．

$$/\chi/_I \geq \mu(\chi)$$

定義 16 プログラムに対する妥当な解釈
プログラム Γ に対する妥当な解釈 I_Γ とは，全てのプログラムの要素の式に対して妥当な解釈を言う．つまり，

$$\forall\chi \in \Gamma,\ /\chi/_{I_\Gamma} \geq \mu_\Gamma(\chi)$$

以後, 妥当解釈とは, プログラムに対する妥当な解釈のことを示すものとする．

定義 17 最小解釈
プログラムに含まれるある式 χ における最小解釈 Mi とは妥当解釈 I_Γ のうちで $/\chi/_{I_\Gamma}$ を最小にするような妥当解釈をいう．

$$Mi^\Gamma(\chi) = \inf_{I_\Gamma}\{/\chi/_T\}$$

定義 18 α- 妥当
プログラム Γ の任意の妥当解釈 I_Γ において，$/\chi/_{I_\Gamma} \geq \alpha(> 0)$ を満足する式を α- 妥当式と呼び，$\Gamma \models_\alpha \chi$ で表す．

妥当式は定理のように帰納的に導出されるものではないが, 与えられたプログラムにおいて妥当である (正しい) と認められる式である．

今，仮に真理値 0 を割り当てられた式が存在したとすると, この割り当ては式に含まれた全てのリテラルに対して任意の解釈を許すことを意味する．これは, 妥当解釈という点からいうと，これらのリテラルを含んだ式がプログラムに存在しないことと同値である．つまり，真理値 0 を割り当てられた式は存在していても，全く意味のないものであるということが出来る．

定義 19 無意味な式
任意のプログラム式において真理値 *0* を割り当てられたものを無意味と呼ぶ．

定理 1 プログラム Γ において，ある式 χ における最小解釈 Mi が存在する時，$\alpha = Mi^\Gamma(\chi)$ とすると χ は α- 妥当式である．

Proof 最小解釈の定義より, 公理系 Γ を満足する任意の解釈 I において $/\chi/_I \geq Mi^{\Gamma}(\chi)$ である.

(Q. E. D)

この定理には次のような意味がある.

最小解釈を求めるということは，その式が取りうる最小の値を求めることであるので，その求められた最小の真理値を式に与えて，プログラムに追加しても矛盾が起こらないということを意味する．つまりある式に 0 より大きい最小解釈が存在するということは，プログラムから新しいしかも正しい知識を得ることが出来たということを意味する.

一方，推論規則から得られた結果に関しては，このような保証があるかどうか分からないので，推論規則が正しい知識を求めることができるかどうかは，前述したとおりその妥当性を調べる必要がある.

普通, 論理における推論規則の正しさは, 完全性と健全性を示すことによって証明される.

完全性 プログラムにおいて, 妥当な式であれば, 推論規則で導き出される.

健全性 プログラムにおいて, 推論規則で導き出された式ならば, 妥当な式である.

第 3 章

ファジィ論理プログラミングにおける推論規則について

本章では，ファジィ論理プログラミングにおける推論に関して，最も重要な役割を果たす含意演算と連言演算について，大きく２つの側面から検証し，Łukasiewicz の含意とŁukasiewicz の積演算が最も適した演算であることを示す．

3.1 推論規則の決定

過去の研究において，多くの推論規則が提案されてきている．ここではまず，本ファジィ論理プログラミングに採用する推論規則を決定する．

推論規則には， modus ponens， modus tollens， syllogism，導出原理など数多く存在する．本論においてこれらの中からファジィ論理プログラミングの推論規則として採用するのは, modus ponens と結合, 述語変数への代入の 3 つである．

これらを選んだ基準は，ファジィ Prolog の実装を行う際に必要となるか，ならないかである．syllogism や分配などの推論規則は, ファジィ Prolog として実装する場合に必要となることはまずないし，導出原理は扱う論理式が節形式ではないので適用出来ない．

modus ponens

前提 ψ, $\phi \leftarrow \psi$ より ϕ を推論する．

結合

前提 $\psi_1, \cdots, \psi_n$ より $\psi_1 \wedge \cdots \wedge \psi_n$ を推論する．

述語変数への代入

前提 $\forall x \chi(x)$ より $\chi(a)$ を推論する．

このうち, 結合は, modus ponens の前件部が複数のリテラルの積項として表わされている場合

に必要な推論規則であるが, 推論結果がプログラム式の形式ではないので, modus ponens に伴って用いられる補助規則とする. したがって, 結合だけを単独で用いることは出来ない.

3.2 推論規則における含意演算と連言演算の満たすべき性質 – 要請 1

ここでは, 推論規則と定理の関係について考える.

定理に関しては, 向殿らは次のような立場を取ることを提案しており [19], ファジィ論理の推論規則を考える上で広く認められている.

要請 1

いかなる解釈に対しても, 論理的帰結 D の真理値は, 前提 C の真理値よりも, 常に, 大きいか, 又は, 等しい. すなわち,

$$T(C) \leq T(D) \tag{3.1}$$

ここで論理的帰結とは推論結果を意味する.

この要請は, 前提 C(公理や定理) から得られる論理的帰結 D(定理) の真理値は, 前提の真理値で下限値が決定されているということを意味しており, 前提によって推論結果の真らしさが保証されていると言う点で, プログラムに与えられる真理値が下限値の意味であることにもマッチし, 人間の感覚に合った自然な要請ということが出来る. この要請は 2 値論理プログラミングの場合も満足している. Lee の理論は, 公理が真理値 0. 5 以上の割り当てをされている時のみしか成り立たないのでこの要請は満足していないが, 向殿らのファジィ導出原理は, この要請を満足している.

ここでは, 先ほどの要請を満足するという立場から, 推論において真理値の計算に関係する演算について考察する.

modus ponens の場合, 要請における前提 C は, ψ と $\phi \leftarrow \psi$ の連言となる. 普通, 連言は演算として考える.

2 つの前提の真理値の連言演算を $*$ と考えると, 要請の式 (3.1) は,

$$\begin{aligned} T(C) &= T(\psi) * T(\phi \leftarrow \psi) \\ &\leq \ T(D) = T(\phi) \end{aligned} \tag{3.2}$$

となる. この時, ファジィ論理プログラミングでは $T(\phi \leftarrow \psi) = T(\phi) \leftarrow T(\psi)$ という立場を取っているので, この場合に要請を満足するかどうかを, 含意演算, 連言演算に関して調べる必要が生

じる. 結合に関しては,

$$
\begin{aligned}
& T(C) = T(\psi_1) \wedge \cdots \wedge T(\psi_n) \\
&= \; T(D) = T(\psi_1 \wedge \cdots \wedge \psi_n)
\end{aligned} \tag{3.3}
$$

とし, 述語変数への代入に関しては,

$$
\begin{aligned}
& T(C) = T(\forall x \chi(x)) \\
&= \; T(D) = T(\chi(a))
\end{aligned} \tag{3.4}
$$

とすれば, 要請を満足しているので考察の必要はない.

一般的には連言演算には min 演算が用いられる. その場合, 要請を満足する含意演算には, 次の例に示すように直観主義論理で用いられる Gödel 含意があるが, Kleene-Dienes や Lukasiewicz などの含意は満足しない.

例 2 Gödel 含意と Kleene-Dienes 含意の場合

- Gödel 含意 ($\leftarrow_{\mathrm{G}}$)

$$
\begin{aligned}
& T(\psi) * T(\phi \leftarrow \psi) \\
& = \min\{T(\psi), (T(\phi) \leftarrow_{\mathrm{G}} T(\psi))\} \\
& = \begin{cases} \min\{T(\psi), 1\} & \leq T(\phi) \quad (\text{if } T(\psi) \leq T(\phi)) \\ \min\{T(\psi), T(\phi)\} & \leq T(\phi) \quad (\text{otherwise}) \end{cases}
\end{aligned}
$$

- Kleene-Dienes 含意

$$
\begin{aligned}
& T(\psi) * T(\phi \leftarrow \psi) \\
& = \min\{T(\psi), \max\{T(\phi), 1 - (\psi)\}\} \\
& = \begin{cases} \min\{T(\psi), T(\phi)\} & \leq T(\phi) \\ & (\text{if } 1 - T(\psi) \leq T(\phi)) \\ \min\{T(\psi), 1 - T(\psi)\} & \not\leq T(\phi) \\ & (\text{otherwise}) \end{cases}
\end{aligned}
$$

そこで, 連言演算を min に限らず, 連言の「かつ」という意味を持つ演算であれば良いと言うように解釈を拡大してみると, t-norm 演算がそれに適合する. そこで, 連言演算を t-norm 演算として, 式 (3.2) を満足する含意演算を考える. その結果は表 3.1の通りである. ただし, 表 3.1の各演算は次の通りである.

$*$ $\leftarrow$	Kl	Re	Łu	Gö	Go	Za
Lo				$\surd$		
Al				$\surd$	$\surd$	
Łu	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
Dr	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$

表 3.1: 要請を満足する演算

定義 20 *[2]*
含意演算 $a \leftarrow b$

Kl: *Kleene-Dienes* $\max\{1-b, a\}$

Re: *Reichenbach* $1-b+a\times b$

Łu: *Łukasiewicz* $\min\{1-b+a, 1\}$

Gö: *Gödel* $\begin{cases} 1 & (\ \mathit{if}\ b \le a) \\ a & (\ \mathit{otherwise}) \end{cases}$

Go: *Goguen* $\begin{cases} 1 & (\ \mathit{if}\ b = 0) \\ \min\{1, \frac{a}{b}\} & (\ \mathit{otherwise}) \end{cases}$

Za: *Zadeh* $\max\{1-b, \min\{a, b\}\}$

連言演算 $a * b$

Lo: *Logical* $\min\{a, b\}$

Al: *Algebraic* $a \times b$

Łu: *Łukasiewicz* $\max\{a+b-1, 0\}$

Dr: *Drastic* $\begin{cases} 0 & (\ \mathit{if}\ a < 1\ \mathit{and}\ b < 1) \\ a & (\ \mathit{if}\ b = 1 \\ b & (\ \mathit{if}\ a = 1) \end{cases}$

例示した Gödel の含意と論理積同様に, 表中 $\surd$ のある組合せでは式 (3.2) が満足されている.

いま, 連言演算は論理積, 代数積, Lukasiewicz 積, 激烈積の順に強くなり, 強い演算ほど演算結果は小さく, 0 になり易くなる. よって, 要請の式がある連言演算で成り立てば, それより強い連言

演算においても必ず成り立つことがいえる．このことから，含意演算を確定した場合に，要請を満足する連言演算が複数存在する場合は，最も弱い連言演算が最も特徴的な連言演算であるということが出来る．

以上の結果より，

- Kleene-Dienes 含意 -Łukasiewicz 積
- Reichenbach 含意 -Łukasiewicz 積
- Łukasiewicz 含意 -Łukasiewicz 積
- Gödel 含意 - 論理積
- Goguen 含意 - 代数積
- Zadeh 含意 -Łukasiewicz 積

が要請を満足する演算の組み合せであるといえる．

3.3 推論規則における含意演算と連言演算の満たすべき性質 – 要請 2

次に定理の立場からではなく，妥当性の立場から推論に関わる演算を考察してみる．ここでは，推論規則によって得られるプログラム式を対象として，前提に矛盾しないような真理値の割り当てを考察する．

推論規則によって得られるプログラム式への妥当な真理値割り当てを求めるために，前提の 2 つの式を真理値に関する制約と見なし，それを解くこととする．簡単に言えば，2 つの前提式を次のような連立方程式に見立てて，その解として推論結果の真理値 $T(\phi)$ を求めるということである．$\alpha = T(\phi)$ とすれは，このことから ϕ が α- 妥当であることがいえる．

要請 2

定理において妥当な解釈は，前提における妥当な解釈と矛盾してはならない．

$$\begin{cases} T(\psi) & = \alpha > 0 \\ T(\phi) \leftarrow T(\psi) & = \beta > 0 \end{cases} \tag{3.5}$$

含意	解	条件
Kleene-Dienes	$T(\phi) = \beta$	$(\beta \geq 1 - \alpha)$
Reichenbach	$T(\phi) = \frac{\alpha+\beta-1}{\alpha}$	$(\alpha + \beta > 1)$
Łukasiewicz	$T(\phi) \geq \alpha$ $T(\phi) = \alpha + \beta - 1$	$(\beta = 1)$ $(\alpha + \beta > 1)$
Gödel	$T(\phi) \geq \alpha$ $T(\phi) = \beta$	$(\beta = 1)$ (otherwise)
Gouguen	$T(\phi) \geq \alpha$ $T(\phi) = \alpha \times \beta$	$(\beta = 1)$ (otherwise)
Zadeh	$T(\phi) = \beta$	$(\alpha \geq \beta \geq 1 - \alpha)$

表 3.2: 含意演算と方程式の解

ただし, 無意味な式の定義により 2 つの前提に与えられる真理値が 0 である場合は存在しないので, どちらも 0 より大きい値であるとした.

先ほどの含意演算について, この方程式を解くと表 3.2のようになる.

例えば, Łukasiewicz の場合, $T(\phi) \leftarrow T(\psi) = \min\{T(\phi) + T(\psi) - 1, 1\}$ より,

- $\beta = 1$

$$\begin{cases} T(\psi) & = & \alpha \\ T(\phi) + T(\psi) - 1 & \geq & 1 \end{cases}$$

よって, $T(\phi) \geq \alpha$

- $\beta \neq 1$

$$\begin{cases} T(\psi) & = & \alpha \\ T(\phi) + T(\psi) - 1 & = & \beta \end{cases}$$

よって, $T(\phi) = \alpha + \beta - 1 > 0$

含意	推論結果の真理値	条件
Kleene-Dienes	$T(\phi) = \beta$	$(\beta \geq 1 - \alpha)$
Reichenbach	$T(\phi) = \frac{\alpha+\beta-1}{\alpha}$	$(\alpha + \beta > 1)$
Łukasiewicz	$T(\phi) = \max\{\alpha + \beta - 1, 0\}$	
Gödel	$T(\phi) = \alpha \wedge \beta$	
Gouguen	$T(\phi) = \alpha \times \beta$	
Zadeh	$T(\phi) = \beta$	$(\alpha \geq \beta \geq 1 - \alpha)$

表 3.3: 含意演算と推論結果の真理値

各条件は，その解を得るための条件であり，条件の範囲に含まれないような場合は，前提への真理値の割り当て自体に誤りがある場合である．
例えば，Kleene-Dienes の場合，$\beta < 1 - \alpha$ とすると，$\max\{T(\phi), 1 - \alpha\} = \beta$ とはなり得ない．

ここで，定義 15，16から式に真理値が割り当てられたとき，その真理値よりも大きくなるような解釈が許されるので，表 3.2の解から，推論結果の真理値は表 3.3のように定義できる．例えば，Łukasiewicz 含意の場合は，定義 19から解の存在しない場合を真理値 0 に割り当てると，

$$\begin{cases} \beta = 1\text{のとき} & T(\phi) \geq \alpha \\ \alpha + \beta - 1 > 0\text{のとき} & T(\phi) = \alpha + \beta - 1 \\ \text{otherwise} & 0 \end{cases}$$

よって，$T(\phi) = \max\{\alpha + \beta - 1, 0\}$ とすることが出来る．

この表 3.3より，適したŁukasiewicz 含意 の対は Łukasiewicz 積に，Gouguen 含意の対は代数積に，Gödel 含意の対は論理積に各々相当し，t-norm で表現可能であることがわかる．

これらの含意は，R- 含意と呼ばれ，その定義式は T を t-norm とすると，

$$a \leftarrow b = \sup\{c \in [0, 1] | T(b, c) \leq a\}$$

であり，まさに要請の式に相当している．

一方，Kleene-Dienes と Zadeh 含意の対は，β のみで表わされ，Reichenbach の含意に対応する適当な t-norm は存在しないということがわかる．これらの結果は，Dubois らの [3] で論じられている pseudo-conjunction に相当する．

これら3つの含意は，S- 含意と呼ばれ，その定義式は S を t-conorm とし，n を否定演算とすると，

$$a \leftarrow b = S(a, n(b))$$

となる．一般的には $n(a) = 1 - a$ である．

pseudo-conjunction は，S- 含意を R- 含意と見なす場合に定義式の t-norm に相当する演算である．しかし，pseudo-conjunction においては交換律 ($f(a,b) = f(b,a)$) が成立しないので t-norm ではない．よってこれらは求める連言演算とはなりえない．

3.4 ファジィ論理プログラミングに適した含意演算と連言演算

ここでは，ファジィ論理プログラミングに適した含意演算と連言演算について考察する．

前の2つの節で検討した推論規則における含意演算と連言演算の満たすべき性質から得られた表3.3と表3.1の結果をともに満足するのが, Łukasiewicz 含意 - Łukasiewicz 積, Gödel 含意 - 論理積, Goguen 含意 - 代数積の3組であるということが出来る．このことからこの3組がファジィ論理プログラミングにおける推論に適した演算であると言うことが出来る．

既に, 提案されているファジィ Prolog を見てみると, M. Van Emden のシステム [4] では, 含意については言及していないが, 連言を代数積で実現していることから, 含意を Gouguen 含意として解釈すれば, Gouguen 含意 - 代数積組であるファジィ論理プログラミングシステムであるということが出来る．また, 菊池らの PROFIL[9] は Gödel 含意 -min 積組である．Łukasiewicz 含意 - Łukasiewicz 積組は F. Klawonn ら [12] においてŁukaisewicz 論理に基づくるエキスパートシステムとして用いられているが，前述の通り，これはファジィ論理プログラミングシステムといえないものである．

本論文では, この3組の中からŁukasiewicz 含意 - Łukasiewicz 積組を採用しているが, その理由は, 次の通りである．

まず, Łukasiewicz 含意以外の含意は, modus tollens を考えると問題が生じる．modus tollens は $a \leftarrow b = \neg b \leftarrow \neg a$(但し, $T(\neg a) = 1 - T(a)$) が満たされる含意では成り立つが, Gödel も Gouguen もこの関係が成り立たない．さらにŁukasiewicz 含意は, 多くの研究で提案された含意演算が満たすべき性質について, 最も多くを満足している．[1]

以上のような理由もあるが, 本論でŁukasiewicz 含意 -Łukasiewicz 積組を採用した最も大きな理由の1つは, 推論の過程で推論結果の真理値が0になる場合があり, 推論を途中で切ることが出来

るということである.

ファジィ論理プログラミングの特徴として, $(0,1]$ の実数値が真理値として公理や定理に与えられることがあるが, 推論の過程において, Gödel 含意 - 論理積, Goguen 含意 - 代数積の組み合わせの場合は, 推論結果の真理値は必ず 0 より大きくなるので, 真理値を無視してプログラム式だけで推論結果を考えると, 2 値論理プログラミングとまったく同じ結果を得ることになる. つまり, 真理値は推論に影響を与えないということになる.

その上ファジィ論理プログラムでは, 2 値の場合では採用されなかった真理値が 1 未満の式がプログラムとして採用されるので, 当然プログラムに含まれる式の数は 2 値の場合よりもかなり増加する. このことは, 推論動作速度の低下にそのままつながってくる.

一方, Łukasiewicz 含意 -Łukasiewicz 積組は推論結果の真理値が 0 になる場合がある. よって, 定義 19よりその推論規則の適用は失敗であったとみなすことが出来き, 無駄な推論動作を節約できる. これは, 2 値論理プログラミングとは異なったファジィ論理プログラミング独特の結果である.

ファジィ論理プログラミングが単に重みを与えた 2 値論理プログラミングではないことを主張する上でも, この性質は重要なものであると言わざるを得ない. なお, この性質は, 再帰的な規則式を扱った場合にも, 推論を途中で停止できるという効果をもつ.

例 3 再帰的な規則式の例

$D(x+1) \leftarrow D(x)$ という規則式を考える. この規則式が表わす意味は, 「x が D であれば, $x+1$ も D である」となる. この規則式に *1* 未満の真理値が与えられるということは, この規則が, 確実ではないということである. 例えば, $D(x)$ としてファジィ述語「距離 x は近い」とするとこの規則式の意味が理解できる. ここで, $T(D(x+1) \leftarrow D(x)) = t < 1$ とする. このとき *D(0)* を事実式として公理に加えた場合を考える.

· *Lukasiewicz*含意 -*Lukasiewicz*積組の場合

$T(D(0)) = 1 > T(D(1)) = t > T(D(2)) = 2t-1 > T(D(3)) = 3t-2 > \cdots > T(D(i)) = it-(i-1) > \cdots$ となる. このとき, $\frac{m-1}{m} > t > \frac{m-2}{m-1}$ なる m が存在して, $T(D(m-1)) = (m-1)t-(m-2) \geq 0 \geq mt-(m-1) = T(D(m))$ となる. よって, この推論は m 回目に *0* になる.

·*Gödel* 含意 - 論理積の場合

$T(D(0)) = 1 > T(D(1)) = t = T(D(2)) = T(D(3)) \cdots$ となる. よって, この推論は無限に繰り返される.

·*Goguen* 含意 - 代数積の場合

$T(D(0)) = 1 > T(D(1)) = t > T(D(2)) = t^2 > T(D(3)) = t^3 > \cdots$ となる. 推論が進むに連れて真理値が小さくなりのだが, $t > 0$ であるので, この推論は無限に繰り返される.

例示したように, Gödel 含意 - 論理積と Goguen 含意 - 代数積の場合は, このような規則式が適用されると, 無限に推論が進んでしまい, 実装した場合は数値の丸め込みで真理値が 0 となって停止するが, 理論上は停止しなくなる.

以上の様な点から, Łukasiewicz 含意 -Łukasiewicz 積がファジィ論理プログラミングに適していると判断できる. 次の章ではこの体系におけるファジィ論理プログラミングを定義し, 推論規則が完全性と健全性を満たすことを示す.

第 4 章

推論規則の完全性と健全性

ここでは，提案する含意演算にŁukasiewicz の含意を，連言演算にŁukasiewicz の積演算を採用したファジィ論理プログラミングの推論規則（以後, 導出規則と呼ぶ）の正しさを示すために，その完全性と健全性を示す.

4.1 導出規則

あらためてファジィ論理プログラミングの推論規則である導出規則について定義する.

定義 21 Łukasiewicz 積

演算 $*$ を *Łukasiewicz* 積とする.

$$\alpha * \beta = \max\{\alpha + \beta - 1, 0\} \tag{4.1}$$

定義 22 Łukasiewicz 含意

演算 $\leftarrow$ を *Łukasiewicz* 含意とする.

$$\alpha \leftarrow \beta = \min\{1 - \beta + \alpha, 1)\} \tag{4.2}$$

定義 23 導出規則

導出規則の適用によって, プログラム Γ から得られるプログラムを Γ' とする.

- *I. modus ponens*
  事実式 ϕ に関して

$$\mu_{\Gamma'}(\phi) = \max\{/\psi/_{\Gamma} * \mu_{\Gamma}(\phi \leftarrow \psi), \mu_{\Gamma}(\phi)\}$$

- *II. substitution*
  $\chi' = \chi\theta$ なる代入 θ に関して

$$\mu_{\Gamma'}(\chi') = \max\{\mu_{\Gamma}(\chi), \mu_{\Gamma}(\chi')\}$$

但し，*1* 回の適用につき，*I*, *II* のいずれか *1* つだけが適用される．それ以外のプログラム式 χ は，$\mu_{\Gamma'}(\chi) = \mu_{\Gamma}(\chi)$ である．

定義 24 補助導出規則
導出規則 *I* を適用する際，次の補助規則を適用することが出来る．

- *I. combination*
  規則式 $\phi \leftarrow \psi_1, \cdots \psi_n$ に関して

$$\mu_{\Gamma'}(\psi_1 \wedge \cdots \wedge \psi_n) = \mu_{\Gamma}(\psi_1) \wedge \cdots \wedge \mu_{\Gamma}(\psi_n)\}$$

定義 25 導出プログラム，最大プログラム
導出規則の有限回の適用によって Γ から導出されるプログラム Γ' を，導出プログラムと呼び $\Gamma \Rightarrow \Gamma'$ と表す．また，

$$Md^{\Gamma}(\chi) = \sup_{\forall \Gamma'}\{\mu_{\Gamma'}(\chi) | \Gamma \Rightarrow \Gamma'\}$$

を式 χ に関する最大プログラムと呼ぶ．なお，始めに与えられるプログラム（公理系） Γ_0 は最大プログラムであると，仮定する．

最大プログラムは，同じ推論結果を得るような複数の推論過程が存在する場合は，その中から最も真に近いものを採用することを意味する．これはプログラム式に与える真理値が下限値であるためで，全ての推論結果の交わりを取った結果である．

例えば，同じ推論結果に対して 0.6, 0.8, 0.7 の三つの結果が得られたとすると，各々の真理値の意味は 0.6, 0.8, 0.7 以上の解釈を許すというものであるので，これら全てを満足する真理値は 0.8 ということになる．

いま，χ に関して $Md^{\Gamma}(\chi) \geq \mu_{\Gamma'}(\chi)$ であるのだから，最大プログラムによって規定される妥当解釈は，最大プログラム以外の導出プログラムにおいても妥当解釈となる訳である．

定義 26 α- **導出**

導出結果 (定理)χ の真理値は $Md^{\Gamma}(\chi)$ で与えられる．また，$Md^{\Gamma}(\chi) \geq \alpha(> 0)$ である時，

$$\Gamma \vdash_{\alpha} \chi$$

で表し，χ はプログラム Γ より α- 導出されたという．

4.2 導出規則の完全性と健全性

ここでは，導出規則の完全性と健全性について考察する．

完全性 プログラム Γ の任意の α- 妥当式 χ は，導出規則により α- 導出される．$(Mi^{\Gamma}(\chi) \leq Md^{\Gamma}(\chi))$

$$\Gamma \models_{\alpha} \chi \Rightarrow \Gamma \vdash_{\alpha} \chi$$

健全性 α- 導出された任意の式 χ は，プログラム Γ の α- 妥当式である．$(Md^{\Gamma}(\chi) \leq Mi^{\Gamma}(\chi))$

$$\Gamma \vdash_{\alpha} \chi \Rightarrow \Gamma \models_{\alpha} \chi$$

これらの証明は以下のようになる．健全性の証明に関しては [12] を参照している．

補題 1 導出規則の I は健全性を満たす．

Proof Łukasiewicz 含意 -Lukasiewicz 積は modus ponens に関して要請を満たすので，事実式 ψ の任意の妥当解釈 T において，定理 1と定義 16より

$$\begin{aligned} &/\phi/_T \\ &\quad \geq \max\{/\psi/_T * /\phi \leftarrow \psi/_T, a(\phi)\} \\ &\quad \geq \max\{\mu_{\Gamma}(\psi) * \mu_{\Gamma}(\phi \leftarrow \psi), \mu_{\Gamma}(\phi)\} \\ &\quad = \mu_{\Gamma'}(\phi) \end{aligned}$$

よって，任意の妥当解釈による値が導出されたプログラム Γ' での値より大きいのだから，$Md^{\Gamma}(\phi) \leq Mi^{\Gamma}(\phi)$

(Q. E. D)

補題 2 導出規則の II は健全性を満たす.

Proof 任意の妥当解釈 T において, 定理 1, 定義 16, 式 (3.4) より

$$
\begin{aligned}
/\chi'/_T &\geq \max\{/\chi/_T, \mu_\Gamma(\chi')\} \\
&\geq \max\{\mu_\Gamma(\chi), \mu_\Gamma(\chi')\} \\
&= \mu_{\Gamma'}(\chi')
\end{aligned}
$$

よって, 補題 1と同様に $Md^\Gamma(\chi') \leq Mi^\Gamma(\chi')$

(Q. E. D)

補題 3 補助導出規則 I は健全性を満たす.

Proof 任意の妥当解釈 T において, 定理 1, 定義 16, 式 (3.3) より

$$
\begin{aligned}
&/\psi_1 \wedge \cdots \wedge \psi_n/_T \\
&\quad = /\psi_1/_T \wedge \cdots \wedge /\psi_n/_T \\
&\quad \geq \mu_\Gamma(\psi_1) \wedge \cdots \wedge \mu_\Gamma(\psi_n) \\
&\quad = \mu_{\Gamma'}(\psi_1 \wedge \cdots \wedge \psi_n)
\end{aligned}
$$

よって, 補題 1と同様に $Md^\Gamma(\psi_1 \wedge \cdots \wedge \psi_n) \leq Mi^\Gamma(\psi_1 \wedge \cdots \wedge \psi_n)$

(Q. E. D)

定理 2 導出規則は健全性を満たす.

Proof 補題 1, 2, 3より明らか.

(Q. E. D)

補題 4 導出規則 I は完全性を満たす.

Proof プログラム Γ の ϕ に関する最大プログラム $M_d^\Gamma(\phi)$ について, 最大プログラムを Γ_m ($\Gamma \Rightarrow \Gamma_m$) とすると, $M_d^\Gamma(\phi) = \mu_{\Gamma_m}(\phi) = /\phi/_I \geq \mu_\Gamma(\phi)$ なる解釈 I を考えることが出来る. この解釈 I は ϕ に対して妥当であるので, $\phi \leftarrow \psi$ に関しても妥当な解釈となることを示せはよい.

導出規則 I より, $M_d^\Gamma(\phi) = \max\{\mu_\Gamma(\phi), \mu_\Gamma(\psi) * \mu_\Gamma(\phi \leftarrow \psi))\}$

$$
\begin{aligned}
& /\phi \leftarrow \psi/_I \\
&= /\phi/_I \leftarrow /\psi/_I \\
&= \mu_{\Gamma_m}(\phi) \leftarrow \mu_\Gamma(\psi) \\
&= \max\{\mu_\Gamma(\phi), \mu_\Gamma(\psi) * \mu_\Gamma(\phi \leftarrow \psi)\} \leftarrow \mu_\Gamma(\psi) \\
&= \max\{\mu_\Gamma(\phi) + 1 - \mu_{\Gamma(\psi)}, (\mu_\Gamma(\psi) * \mu_\Gamma(\phi \leftarrow \psi)) + 1 - \mu_\Gamma(\psi), 0\} \\
&= \max\{\mu_\Gamma(\phi) + 1 - \mu_\Gamma(\psi), \mu_\Gamma(\phi \leftarrow \psi), 1 - \mu_\Gamma(\psi)\} \\
&\geq \mu_\Gamma(\phi \leftarrow \psi)
\end{aligned}
$$

以上の結果から解釈 I は妥当解釈となることが言える. よって, $M_d^\Gamma(\psi) = /\psi/_I \geq M_i(\psi)$

(Q. E. D)

補題 5 導出規則 II は完全性を満たす.

Proof　プログラム Γ の χ' に関する最大プログラム $M_d^\Gamma(\chi')$ について, 最大プログラムを Γ_m ($\Gamma \Rightarrow \Gamma_m$) とすると, $M_d^\Gamma(\chi') = \mu_{\Gamma_m}(\chi') = /\chi'/_I \geq \mu_\Gamma(\chi')$ なる解釈 I を考えることが出来る。解釈 I は χ' に関して妥当であるので, プログラム Γ に関して妥当な解釈となることを示せばよい.

$\chi' = \phi$ の場合, $/\chi'/_I = /\phi/_I \geq \mu_\Gamma(\phi)$ であるので, I は妥当解釈となる.
$\chi' = \phi \leftarrow \psi$ の場合, $\mu_\Gamma(\chi') = 1$ の時は $/\chi'/_I = 1$ なので, I は妥当解釈となる.
$\mu_\Gamma(\chi') \neq 1$ の時は $M_d^\Gamma(\chi') \neq 1$ となるので,

$$T(\chi') = T(\phi \leftarrow \psi) = T(\phi) - T(\psi) + 1$$

よって,

$$/\chi'/_I = /\phi/_I - /\psi/_I + 1 \geq \mu_\Gamma(\chi') = \mu_\Gamma(\phi) - \mu_\Gamma(\psi) + 1$$

ここで, $\delta = \chi'/_I - \mu_\Gamma(\chi') \geq 0$ とすると, 解釈 I を $/\phi/_I = \mu_\Gamma(\phi) + \delta$, $/\psi/_I = \mu_\Gamma(\psi)$ としても矛盾しない. この解釈 I は ϕ, ψ に関しても妥当であるので, 解釈 I は妥当解釈となる.

(Q. E. D)

定理 3 導出規則は完全性を満たす.

Proof 補助導出規則 I は解釈を制限するものではないので, 導出規則 I, II が完全であることを示せば十分である. よって, 補題 4, 5より導出規則は完全であることがいえる.

(Q. E. D)

以上から，導出規則の完全性と健全性を示すことが出来た.

例 4 ファジィ論理プログラミングの例
プログラム Γ が,

$$
\begin{array}{rll}
i\ . & \mu_\Gamma(decent(x) \leftarrow honest(x)) & = 1 \\
ii\ . & \mu_\Gamma(decent(x) \leftarrow polite(x)) & = 1 \\
iii\ . & \mu_\Gamma(honest(tom)) & = 0.9 \\
iv\ . & \mu_\Gamma(polite(tom)) & = 0.8 \\
v\ . & \mu_\Gamma(honest(bob)) & = 0.9 \\
vi\ . & \mu_\Gamma(polite(bob)) & = 0.85
\end{array}
$$

であたえられている．このとき， i に導出規則 II を適用して x に tom を代入して，その結果と iii に導出規則 I を適用すると $\mu_\Gamma(decent(tom)) = 0.9$ が得られる.

また， ii に導出規則 II を適用して x に tom を代入して，その結果と iv に導出規則 I を適用すると $\mu_\Gamma(decent(tom)) = 0.8$ が得られる.

よって， $decent(tom)$ の最大プログラム Γ' は $\mu_{\Gamma'}(decent(tom)) = \max\{0.9, 0.8\} = 0.9$ である.

同様に $decent(bob)$ の最大プログラム Γ'' は $\mu_{\Gamma''}(decent(bob)) = \max\{0.9, 0.85\} = 0.9$ となる.

次の章では，ファジィ論理プログラミングの表現能力を高めるために言語ヘッヂと呼ばれる 1 項論理演算を導入し，導出規則を拡張する.

第 5 章

言語ヘッヂを導入したファジィ論理プログラミングについて

言語ヘッヂは，ファジィ集合によってモデル化された曖昧な概念，例えば「若い」などに作用する修飾詞であり，「非常に」や「多少」，「本質的に」などが知られている．この言語ヘッヂを導入することにより，より多様な曖昧な概念を扱うことが可能になった．

言語ヘッヂには大きく分けて，単純な概念を現わすファジィ集合に作用するタイプI（例えば「非常に」）と凸結合などを用いているような複雑な概念を現わすファジィ集合に作用するタイプII（例えば「本質的に」）の２つのカテゴリーが存在する．[32]

本論で扱うのは１階ファジィ述語論理への導入が可能なタイプIの言語ヘッヂであり，１項論理演算としてファジィ論理プログラミングに導入する．

また，プログラム式は否定論理演算を含むことを認めていない．これは，推論規則の完全性，健全性を保つために必要なことであるが，ファジィ論理プログラミングを実用的に用いるためには否定の概念も必要となってくる．そこで本章ではさらに，言語ヘッヂの１つとして否定の概念を導入することを考察する．

5.1 言語ヘッヂの定義

タイプIの言語ヘッヂには， very, exact, much, slightly などがあるが，この内１階ファジィ述語論理へ導入可能な１項論理演算として定義出来るのは， very, more or less, exact そして人工的言語ヘッヂである plus, minus である． much や slightly はファジィ述語のとる項同士の間に成り立つ関係などを用いたりするため，１階ファジィ述語論理の範囲で扱うことが出来ない．また，否定の概念を現わす not も１項論理演算として定義する．

まず，言語ヘッヂが１項論理演算であるために次の条件が必要となる．

定義 27 言語ヘッヂ $\langle h \rangle$ は $[0,1]$ から $[0,1]$ への全射の *1* 変数関数 f として表現できる.

$$T(\langle h \rangle \chi) = f(T(\chi))$$

ここで本論で取り扱う言語ヘッヂを定義する.

very($\langle very \rangle$) はファジィ集合演算の集中化 (CON) によって定義される.

定義 28 very(非常に)

$$T(\langle very \rangle \chi) = CON(T(\chi)) = (T(\chi))^2$$

more or less($\langle morl \rangle$) はファジィ集合演算の拡大化 (DIL) によって定義される.

定義 29 more or less(やや)

$$T(\langle morl \rangle \chi) = DIL(T(\chi)) = (T(\chi))^{0.5}$$

exact($\langle exact \rangle$) はファジィ集合のサポートを与えるような演算によって定義される. サポートとは, ファジィ集合を 1 で α カットした結果をいう.

定義 30 exact(厳密に)

$$T(\langle exact \rangle \chi) = \begin{cases} 1 & (if\, T(\chi) = 1) \\ 0 & (otherwise) \end{cases}$$

plus($\langle plus \rangle$) は強調演算子, minus($\langle minus \rangle$) は弛緩演算子と呼ばれ, それぞれ very と more or less よりも小さい効果をもたらす.

また [32] によれば, これらの間には近似的に

$$T(\langle plus \rangle \langle plus \rangle \chi) = T(\langle minus \rangle \langle very \rangle \chi)$$

という関係が成り立つ必要があり, このことから $T(\langle plus \rangle \chi) = (T(\chi))^{\sqrt{5}-1}$, $T(\langle minus \rangle \chi) = (T(\chi))^{3-\sqrt{5}}$ と定義されている. しかし本論では, 言語ヘッヂを組み合わせたときの性質が, 議論の対象となるので, さらに

$$T(\langle plus\rangle\langle minus\rangle\chi) = T(\chi)$$

という条件を加えた．これにより $\langle plus\rangle$, $\langle minus\rangle$ は次のように定義される．

定義 31 plus

$$T(\langle plus\rangle\chi) = (T(\chi))^{\sqrt[3]{2}}$$

定義 32 minus

$$T(\langle minus\rangle\chi) = (T(\chi))^{\frac{1}{\sqrt[3]{2}}}$$

not($\langle not\rangle$) は否定論理演算と同様に定義する．

定義 33 not(ではない)

$$T(\langle not\rangle\chi) = 1 - T(\chi)$$

以上，6 つの言語ヘッヂを基本ヘッヂと呼ぶ．

これらの基本ヘッヂを組み合わせることによって，可算無限個の合成言語ヘッヂを作ることが可能となる．

定義 34 合成ヘッヂ

言語ヘッヂを含まない論理式 χ に *1* つ以上の基本ヘッヂ *(*$\langle h_1\rangle, \langle h_2\rangle, \cdots, \langle h_l\rangle$*)* が作用するとき，これらのヘッヂをまとめたものを合成ヘッヂ *(*$\langle H\rangle$*)* と呼ぶ．

$$\langle h_1\rangle\langle h_2\rangle\cdots\langle h_l\rangle\chi = \langle H\rangle\chi$$

以後，基本ヘッヂと合成ヘッヂ，さらに $f(x) = x$ の恒等関数で表される恒等ヘッヂ (ヘッヂがない状態と等しい) をまとめてヘッヂと呼ぶものとする．

言語ヘッヂの性質を知るためには全てのヘッヂについてを考察する必要はなく，いくつかの性質ごとに考察すればよい．

定義 35 指数型ヘッヂ

ヘッヂ $\langle H\rangle$ が $T(\langle H\rangle\chi)=(T(\chi))^{\gamma}$ (γ は 0 より大きい実数値) であるようなとき，指数型ヘッヂと呼ぶものとする．特に，$\gamma>1$ である $\langle H\rangle$ を強調化ヘッヂ，$\gamma<1$ である $\langle H\rangle$ を弛緩化ヘッヂと呼ぶ．

定義 36 単射ヘッヂ

ヘッヂ $\langle H\rangle$ が $T(\langle H\rangle\chi)=f(T(\chi))$ であり，さらに f が全単射関数であるとき，$\langle H\rangle$ を単射ヘッヂと呼ぶ．

指数型ヘッヂは単射ヘッヂである．また，全単射関数には逆関数が存在することから次のような補ヘッヂを定義できる．

定義 37 補ヘッヂ

単射ヘッヂ $\langle H\rangle$ が $T(\langle H\rangle\chi)=f(T(\chi))$ であるとき，f の逆関数 f^{-1} に基づいて $T(\langle H^c\rangle\chi')=f^{-1}(T(\chi'))$ なる単射ヘッヂが定義できる．このとき，$\langle H^c\rangle$ を $\langle H\rangle$ の補ヘッヂと呼ぶ．

基本ヘッヂにおいては，$\langle very\rangle$ と $\langle morl\rangle$，$\langle plus\rangle$ と $\langle minus\rangle$ がそれぞれ互いに補ヘッヂの関係になる．

定義 38 単調ヘッヂ

ヘッヂ $\langle H\rangle$ が $T(\langle H\rangle\chi)=f(T(\chi))$ であり，さらに f が $x\leq y$ ならば $f(x)\leq f(y)$ であるとき，$\langle H\rangle$ を単調ヘッヂと呼ぶ．

指数型ヘッヂと $\langle exact\rangle$ は単調ヘッヂである．

5.2 ヘッヂを持つファジィ論理

ファジィ論理は Kleene 代数のモデルであり，次の公理が成立する．

(1) $a\vee b=b\vee a,\ a\wedge b=b\vee a$ 交換律

(2) $a\vee(b\vee c)=(a\vee b)\vee c$

$a\wedge(b\wedge c)=(a\wedge b)\wedge c$ 結合律

(3) $a\vee(a\wedge b)=a$

$a \wedge (a \vee b) = a$　　吸収律

(4)　$a \wedge a = a,\ a \vee a = a$　　べき等律

(5)　$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$　　分配律

(6)　$\neg(a \wedge b) = \neg a \vee \neg b$

$\neg(a \vee b) = \neg a \wedge \neg b$　　DeMorgan 律

(7)　$\neg(\neg a) = a$　　２重否定

(8)　$0 \wedge a = 0,\ 0 \vee a = a$　　最小元

(9)　$1 \wedge a = a,\ 1 \vee a = 1$　　最大元

(10)　$(a \wedge \neg a) \vee (b \vee \neg b) = (b \vee \neg b)$

$(a \wedge \neg a) \wedge (b \vee \neg b) = (a \wedge \neg a)$　　Kleene 律

さらに，Łukasiewicz の含意を導入することにより，

(11)　$a \leftarrow b = \neg b \leftarrow \neg a$　　対偶

(12)　$a \leftarrow a = 1$

(13)　$(a \leftarrow b) \leftarrow c = (a \leftarrow c) \leftarrow b$

(14)　$(a \leftarrow b) \leftarrow a = 1$

となる．これらは古典論理における公理に対応している．

さらに，言語ヘッヂを導入すると，次のような新たな公理が得られる．ただし，$*$ は任意の指数型ヘッヂを，$+$ は任意の強調化ヘッヂを，$-$ は任意の弛緩化ヘッヂを各々表わすのものとする．

(15)　$*_1 *_2 a = *_2 *_1 a$　　指数型ヘッヂの交換律

(16)　$*(a \wedge b) = *a \wedge *b$

$*(a \vee b) = *a \vee *b$

(16′)　$\langle exact \rangle (a \wedge b) = \langle exact \rangle a \wedge \langle exact \rangle b$

$\langle exact \rangle (a \vee b) = \langle exact \rangle a \vee \langle exact \rangle b$

(17)　$-a \leftarrow a = 1$

(18) $a \leftarrow +a = 1$

(18′) $a \leftarrow \langle exact \rangle a = 1$

(19) $\langle very \rangle a = \langle plus \rangle \langle plus \rangle \langle plus \rangle a$

(20) $\langle h \rangle \langle h^c \rangle a = a$ (但し，$\langle h^c \rangle$は単射ヘッヂ$\langle h \rangle$の補ヘッヂ)

これ以外にも公理が存在すると思われるが，言語ヘッヂとŁukasiewicz 含意を持つファジィ論理の公理系に関する研究は本論文の対象ではないので，ここでは議論しない．なお，公理 (11) ～ (20) の証明は付録として掲載してある．本章での興味の対象は，これらを用いて言語ヘッヂを持つファジィ論理プログラミングの推論規則を定義することにある．

5.3 言語ヘッヂを持つファジィ論理プログラミングの推論規則

ここでは，言語ヘッヂを導入した場合のファジィ論理プログラミングの推論規則について考察する．対象となる言語ヘッヂを，指数型ヘッヂ，単調ヘッヂ，単射ヘッヂとし，各々について考察する．

まず，プログラム式の定義を拡張する．

定義 39 ヘッヂを持つプログラム式

- $Q\langle h \rangle \phi \leftarrow \langle h_1 \rangle \psi_1 \wedge \cdots \wedge \langle h_m \rangle \psi_m$
- $Q\langle h \rangle \psi$

ただし，ϕ，ψ_i は一階ファジィ述語を，Q は束縛子 ($\forall$ もしくは $\exists$) を，$\langle h \rangle, \langle h_1 \rangle, \cdots, \langle h_m \rangle$ はヘッヂを表すものとする．

言語ヘッヂの付く場合も以後，変数が明らかな場合は束縛子を省略して表記することとする．

ここで注意する点は，$Q\langle h \rangle (\phi \leftarrow \psi_1 \wedge \cdots \wedge \psi_m)$ は上の式の形式展開できないので，プログラム式とはならないという点である．このことから，ヘッヂは式中のリテラルに対してのみ適用できるということがいえる．

5.3.1 指数型ヘッヂ

指数型ヘッヂ $\langle h \rangle$ は $T(\langle h \rangle \psi) = (T(\psi))^\gamma$ (γ は 0 より大きい実数値) で表すことができる．この時次のことがいえる．

補題 6 $T(\langle h\rangle\psi) = (T(\psi))^{\gamma}$ なる指数型ヘッヂ $\langle h\rangle$ について，プログラム Γ で $\langle h\rangle\psi$ の最小解釈が $Mi^{\Gamma}(\langle h\rangle\psi) = \beta$ であるとすると，ψ の最小解釈は $Mi^{\Gamma}(\psi) = \beta^{1/\gamma}$ となる．

Proof x^{γ} $(x \in [0,1])$ は x に関する全単射関数であるので $T(\langle h\rangle\psi)$ から $T(\psi)$ を一意に求めることが出来る．また，x に関して単調性を満たしているので，$/\langle h\rangle\psi/_I \geq \beta$ であるような任意の妥当解釈 I において $/\psi/_I \geq \beta^{1/\gamma}$ が成り立つ．よって，ψ の最小解釈は $Mi^{\Gamma}(\psi) = \beta^{1/\gamma}$ となる．

(Q. E. D)

この証明から分かるように，単調ヘッヂでかつ単射ヘッヂであるようなヘッヂにおいても，同様のことがいえるが，本論で扱うヘッヂにおいて，このようなヘッヂは指数型ヘッヂだけである．($\langle not\rangle$ は単調ヘッヂでなく $\langle exact\rangle$ は単射ヘッヂでない．)

ここで次のような推論規則が定義する．

定義 40 拡張導出規則

導出規則 I を適用する際，次の補助規則を適用することが出来る．

ψ に関して，$\langle h\rangle$ を $T(\langle h\rangle\psi) = (T(\psi))^{\gamma}$ である指数型ヘッヂとするとき，

- *I. exponential hedge I.*

  規則式 $\phi \leftarrow \psi_1, \cdots, \langle h\rangle\psi_i, \cdots, \psi_n$ に関して

$$\mu_{\Gamma'}(\psi_1 \wedge \cdots \wedge \langle h\rangle\psi_i \wedge \cdots \wedge \psi_n) = \min\{\mu_{\Gamma}(\psi_1), \cdots, (\mu_{\Gamma}(\psi_i))^{\gamma}, \cdots, \mu_{\Gamma}(\psi_n)\}$$

- *II. exponential hedge II.*

$$\mu_{\Gamma'}(\psi_1 \wedge \cdots \wedge \psi_i \wedge \cdots \wedge \psi_n) = \min\{\mu_{\Gamma}(\psi_1), \cdots, (\mu_{\Gamma}(\langle h\rangle\psi_i))^{1/\gamma}, \cdots, \mu_{\Gamma}(\psi_n)\}$$

ここで問題となるのが，推論過程で再帰的に同じリテラルが現われる場合である．ここでいう同じリテラルとは同一の命題という意味でる．取る項の異なる述語は異なる命題であるので同じリテラルではない．

2 値論理プログラミングの場合も推論過程で再帰的に同じリテラルが現われる場合は，推論が無限に繰り返されてしまうので問題があるが，完全性や健全性には特に影響はない．言語ヘッヂ

を認めないファジィ論理プログラミングでも同様である．しかし，言語ヘッヂを導入する場合は大きな問題となる．

例えば，ファジィ論理プログラミングにおいて $A \leftarrow B$ という式がプログラム Γ に含まれているとする．問題が起こるのはこの B の推論過程に A が現われるときである．

導出規則の健全性は，B に任意の解釈 I を与えても $/A \leftarrow B/_I \geq \beta$ の成り立つことを前提に証明されている．これは B の真理値が A に関係なく決められる場合に成り立つが，B の真理値が A の真理値に影響される場合は一般にはいえない．

例えば強調ヘッヂ $T(\langle h \rangle A) = (T(A))^\gamma$，$\gamma > 1$ のような値を小さくする効果をもつヘッヂ $\langle h \rangle$ において $\mu_\Gamma(\langle h \rangle A \leftarrow B) = \beta$，$\beta < 1$ のようなプログラムがあるとすると問題が生じる．

B の真理値は A の真理値に影響されると仮定すると，$T(B) = f(T(A))$ と表わせる．B は A から推論されるので，導出規則より明らかに $f(T(A)) \leq T(A)$ である．

このとき $T(\langle h \rangle A \leftarrow B) = \min\{(T(A))^\gamma - f(T(A)) + 1\}$ が $T(A)$ だけで決定されて，しかも 1 より小さくなる場合，導出規則の健全性が成り立たなくなることがある．

$T(\langle h \rangle A \leftarrow B)$ は必ず $[0, 1]$ の値を取る連続の関数であるので，必ず最小値 ν が存在する．もし，プログラム $\Gamma \mu_\Gamma(\langle h \rangle A \leftarrow B) = \beta > \nu$ と定義されていると，B に $T(\langle h \rangle A \leftarrow B)$ を最小にするような解釈 I を与えたとき，$/A \leftarrow B/_I = \nu \ngeq \beta$ となってしまう．

γ が $0 < \gamma \leq 1$ であれば，$(T(A))^\gamma - f(T(A)) + 1$ は常に 1 より大きいので，任意の β と任意の解釈 I について $/\langle h \rangle A \leftarrow A)/_I \geq \beta$ が成り立つ．

この問題は指数型ヘッヂだけでなくすべてのヘッヂにも共通の問題である．そこでこの問題を回避するために，次の様にファジィ論理プログラミングに制限を加えることとする．

再帰的規則に対する制限

プログラム Γ に $\langle h \rangle \phi \leftarrow \psi$ または $\phi \leftarrow \psi$ が含まれるとき，サブプログラム Γ_ψ に ϕ が含まれてはならない．

言語ヘッヂのないファジィ論理プログラミングではこの様な制限は必要ないが，この制限に言語ヘッヂのない場合も含めたのは，無意味な推論が無限に繰り返されてしまう問題も合わせて排除するためである．意味のある推論を行う上ではこの制限はさほど大きなものではないといえる．

再帰的規則に対する制限が与えられているので，補題 6の証明は拡張導出規則 I，II の健全性の証明そのものであるので，

補題 7 拡張導出規則 *I*，*II* は健全性を満たす．

Proof　補題6の証明より明らかである．

(Q. E. D)

さらに，

補題 8 拡張導出規則 *I*, *II*は完全性を満たす．

Proof　拡張導出規則 I，II は補助導出規則である combination と同様に，新たに解釈を制限するものではないので，明らかである．

(Q. E. D)

指数型ヘッヂを規則の頭部に適用すると，規則に重用度を与えることが出来るようになる．

規則に真理値を与える場合は，推論結果に関して常に真理値を小さくするようにしか重要度を与えられないが，強調化ヘッヂを頭部に適用すると，次の例のように真理値を大きくする効果が得られ，その規則によって得られる結果が他のヘッヂの付いていない同じ頭部を持つ規則の結果よりも，推論結果に関して重要であるということを示すことが出来る．また，弛緩化ヘッヂを用いれば逆の効果が得られる．

例 5 指数型ヘッヂによる重要度

プログラム Γ が，

$$
\begin{aligned}
\mu_\Gamma(decent(x) \leftarrow honest(x)) &= 1 \\
\mu_\Gamma(decent(x) \leftarrow polite(x)) &= 1 \\
\mu_\Gamma(honest(tom)) &= 0.9 \\
\mu_\Gamma(polite(tom)) &= 0.8 \\
\mu_\Gamma(honest(bob)) &= 0.9 \\
\mu_\Gamma(polite(bob)) &= 0.85
\end{aligned}
$$

であるとき，*tom* と *bob* の *decent* の度合は共に *0.9* であるが，*2* 番目の規則の重要度を上げるために，

$$
\mu_\Gamma(\langle very \rangle decent(x) \leftarrow polite(x)) = 1
$$

とすると，$\sqrt{0.85} \simeq 0.92, \sqrt{0.8} \simeq 0.89$ であることから，$decent(tom)$ は *0.9*，$decent(bob)$ は *0.92* となる．

この結果は，*2* 番目の規則に $\langle very \rangle$ の重要度が与えられたことによるものである．

5.3.2 単調ヘッヂ

ここで考察する単調ヘッヂは単射ではないものとする．基本ヘッヂでは $\langle exact \rangle$ が単調ヘッヂである．

単調ヘッヂは補ヘッヂを持たないので，拡張導出規則 II のような推論規則を定義できない．また，単調ヘッヂが作用しているリテラルに対しても，ヘッヂの値域を超えるような真理値を与えることが出来ない．

これらの問題を解決するために，次の制限を設ける．

単調ヘッヂに対する制限

単調ヘッヂは，プログラム (規則) の頭部に適用できないものとする．

この制限により，単調ヘッヂが作用しているリテラルに対して直接真理値を与えることが出来無くなるので値域を超えるような真理値を与えることも避けられ，単調ヘッヂに関しては拡張導出規則 II の必要性も無くなる．

あとは単調ヘッヂが作用した結果において，妥当解釈に制限が生じないことが明らかになれば，拡張導出規則 I と同様の推論規則を適用しても健全性と完全性を満足することがいえる．

命題 1 プログラム Γ において $\mu_\Gamma(\psi) = \alpha$，$\mu_\Gamma(\phi \leftarrow \psi) = \beta$ である場合，ψ へ与えられる真理値 α が確定値であったとしても，ϕ に関しては，$/\psi/_I = \alpha$ と $/\phi/_I \geq \alpha * \beta$ を満たす任意の解釈が妥当解釈となる．

Proof　いま $/\psi/_I = \alpha$，$/\phi/_I \leftarrow /\psi/_I \geq \beta$ なる任意の解釈 I を考える．

このとき $/\phi/_I \geq \alpha * \beta = \max\{\alpha + \beta - 1, 0\}$ が妥当解釈であることをいえばよい．

$$
\begin{aligned}
/\phi/_I \leftarrow /\psi/_I & \\
\geq & \min\{\max\{\alpha + \beta - 1, 0\} - \alpha + 1, 1\} \\
= & \min\{\max\{\beta, 1 - \alpha\}, 1\} \\
= & \max\{\beta, 1 - \alpha\} \geq \beta
\end{aligned}
$$

よって，解釈 I は妥当解釈である．

(Q. E. D)

この命題は，規則の本体が取り得る真理値がある 1 つの値に制限されてもそこから得られる推論結果の妥当解釈には影響しないことを意味し，単調ヘッヂが規則の本体に含まれていても妥当解釈には影響を与えないことを保証するものである．

次のように推論規則を定義すれば，完全性と健全性を満足する．

定義 41 拡張導出規則

ψ に関して，$\langle h\rangle$ を $T(\langle h\rangle\psi) = f(T(\psi))$ である単調ヘッヂとするとき，

- *III. monotonic hedge*
  規則式 $\phi \leftarrow \psi_1, \cdots, \langle h\rangle\psi_i, \cdots, \psi_n$ に関して

$$\mu_{\Gamma'}(\psi_1 \wedge \cdots \wedge \langle h\rangle\psi_i \wedge \cdots \wedge \psi_n) = \min\{\mu_\Gamma(\psi_1), \cdots, f(\mu_\Gamma(\psi_i)), \cdots, \mu_\Gamma(\psi_n)\}$$

$$\mu_{\Gamma'}(\langle h\rangle\psi) = f(\mu_\Gamma(\psi))$$

補題 9 拡張導出規則 *III* は健全性を満たす．

補題 10 拡張導出規則 *III* は完全性を満たす．

現在，単調ヘッヂの基本ヘッヂは $\langle exact\rangle$ のみであるが，真理値が閾値以上の場合は 1 で閾値未満は 0 とするような閾値関数によって定義されるヘッヂも考えることが出来る．これは，ファジィ集合論における α カットであり，石塚らによる Prolog-Elf[6] で提案された閾値の概念に相当する．

α-threshold($\langle thred_\alpha\rangle$) はファジィ集合の α カットを与えるような演算によって定義される．α-cut とすると，Prolog で用いられるカットオペレータと混同し易いので α-threshold と呼ぶことにする．

定義 42 α-threshold

$$T(\langle thred_\alpha\rangle\chi) = \begin{cases} 1 & (\mathit{if}\ T(\chi) \geq \alpha) \\ 0 & (otherwise) \end{cases}$$

他にも，正数スカラー倍によって定義されるヘッヂ等が考えられる．

5.3.3 その他のヘッヂ

ここで考察するヘッヂは単調性を満たさない単射ヘッヂや単調でも単射でもないようなヘッヂである．基本ヘッヂでは $\langle not\rangle$ がこれに分類される．

単射ヘッヂは補ヘッヂを持つが，指数型ヘッヂのように単調性を満たさないので，補題6のようなことがいえず，前述の拡張導出規則のような推論規則を定義しても，完全性と健全性は満足できない．単調でも単射でもないようなヘッヂに関しても同様である．

例えば，プログラムに young という述語を頭部に持つ規則がいくつかあり，その中に基本ヘッヂ $\langle not\rangle$ を持つものがあったとする．今， young を得るような推論課程と $\langle not\rangle$young を得るような推論課程によって得られる推論結果が， young が 0.8， $\langle not\rangle$young が 0.4 であったとすると，$/young/_I \geq 0.8$， $/\langle not\rangle young/_I \geq 0.4$ が両方を満足する解釈 I となる．しかし， $/\langle not\rangle young/_I = 1 - /young/_I$ より， $/young/_I \leq 0.6$ となり， $/young/_I \geq 0.8$ と矛盾するのでこのような解釈は存在しない．

このような問題を避けるためには単調ヘッヂと同様に，これらのヘッヂが作用しているリテラルに対して直接真理値を与えないという方法が考えられる．しかしそれでも単調性がないため，一般的にはヘッヂが作用しているリテラルに対して α- 妥当性を示すことは出来ない．これまでのヘッヂのような推論規則では健全性と完全性を示すことは難しい．

ここでは，ファジィ論理プログラミングにおいて不可欠と思われる基本ヘッヂの $\langle not\rangle$ について考察する．

まず，先ほど例示した問題を避けるために，単調に対するヘッヂと同様の制限を与える．

$\langle not\rangle$ に対する制限 I

$\langle not\rangle$ は，プログラム (規則) の頭部に適用できないものとする．

$\langle not\rangle$ の場合，プログラム Γ における ψ に関する最小解釈 $Mi_\Gamma(\psi)$ である解釈 T において， $/\langle not\rangle\psi$ は任意の妥当解釈 I に関して，

$$/\langle not\rangle\psi/_T = 1 - Mi_\Gamma(\psi) \geq /\langle not\rangle\psi/_I$$

となってしまう．結局 $\langle not\rangle\psi$ には， ψ に関する最小解釈に矛盾しないように，下限値で与えられるような真理値を与えることは出来ない．

唯一矛盾をきたさない方法は， $\langle not\rangle$ が作用するリテラルに関しては最小解釈以外の解釈を認めないという方法である．つまり，この場合だけ真理値を確定値として与えるということである．

当然，$\langle not \rangle$ が作用するリテラル ψ に関しては妥当な解釈は唯一となるので，ψ を推論結果とするような推論課程に関係する全てのリテラルに対しての妥当解釈に制限が生じることになる．さらにそれらのリテラルを含むプログラム式に制限を加える，というように最終的にはプログラム全体に影響を及ぼすことになる．

しかしプログラム Γ において，$\langle not \rangle\psi$ を本体に含むプログラム式が存在する場合，ψ を推論結果とする推論課程で与えられるような妥当解釈への制限が，その他の推論結果へ影響しなければプログラム全体への影響は避けられる．

いま，プログラム Γ と，式 ψ に関する Γ のサブプログラム Γ_ψ に関して，Γ から推論可能な推論結果の集合を Γ^* と，Γ_ψ から推論可能な推論結果の集合を Γ^*_ψ とする．このとき $\Gamma_\psi \subset \Gamma$ であるので $\Gamma^*_\psi \subset \Gamma^*$ は明らかである．また，Γ_ψ は式 χ に関するサブプログラムであるので，χ は Γ_ψ より推論できる．この推論結果を $\mu_{\Gamma_\chi}(\psi)$ とする．

ここで $B \subset A$ なる集合 A，B において，A から B の要素を全て削除したものを $A - B$ で表わすとする．このとき明らかに，$(\Gamma - \Gamma_\psi)^* \subseteq (\Gamma^* - \Gamma^*_\psi)$ が成り立つ．

このとき，$(\Gamma^* - \Gamma^*_\psi) \cup \{\mu_{\Gamma_\psi}(\psi)\}$ の全ての要素が $(\Gamma - \Gamma_\psi) \cup \{\mu_{\Gamma_\psi}(\psi)\}$ から推論可能であるとき，つまり $(\Gamma^* - \Gamma^*_\psi) \cup \{\mu_{\Gamma_\psi}(\psi)\} = ((\Gamma - \Gamma_\psi) \cup \{\mu_{\Gamma_\psi}(\psi)\})^*$ であるとき，Γ_ψ は Γ から分離可能であると呼ぶものとする．

ψ に関するサブプログラム Γ_ψ がプログラム Γ より分離可能であれば，Γ_ψ における妥当解釈に制限が与えられても，その影響は $(\Gamma^* - \Gamma^*_\psi)$ においては ψ の妥当解釈に与えられる制限だけを考慮すればよいことになる．

このことから，$\langle not \rangle$ にはつぎのような制限を与えれば完全性と健全性についての議論が可能となる．

$\langle not \rangle$ に対する制限 II
プログラム Γ において，$\langle not \rangle\psi$ を本体に含むプログラム式が存在する場合，ψ に関するサブプログラム Γ_ψ が Γ より分離可能でなければならない．

命題 2 プログラム Γ において，ψ に関するサブプログラム Γ_ψ に現れる ψ 以外のリテラルが $\Gamma - \Gamma_\psi$ に含まれていないとき，Γ_ψ は Γ から分離可能である．

Proof Γ_ψ に含まれる ψ 以外のリテラルが $\Gamma - \Gamma_\psi$ に含まれていないことより，$\Gamma^* - \Gamma^*_\psi$ の要素を推論する推論過程でそれらのリテラルが含まれるのは，推論過程に ψ が含まれているときだけである．よって，$\Gamma^* - \Gamma^*_\psi$ は $(\Gamma - \Gamma_\psi) \cup \{\mu_{\Gamma_\psi}(\psi)\}$ から推論される推論結果の集合に含まれる

ので，$(\Gamma^* - \Gamma^*_\psi) \cup \{\mu_{\Gamma_\psi}(\psi)\} \subseteq ((\Gamma - \Gamma_\psi) \cup \{\mu_{\Gamma_\psi}(\psi)\})^*$ となる．いま，$\mu_{\Gamma_\psi}(\psi)$ は Γ の推論結果でもあるので，$\Gamma^* \cup \{\mu_{\Gamma_\psi}(\psi)\} = (\Gamma \cup \{\mu_{\Gamma_\psi}(\psi)\})^*$ である．一般に $(\Gamma - \Gamma_\psi)^* \subseteq (\Gamma^* - \Gamma^*_\psi)$ が成立するので，

$$\begin{aligned}((\Gamma \cup \{\mu_{\Gamma_\psi}(\psi)\}) - \Gamma_\psi)^* &\subseteq ((\Gamma \cup \{\mu_{\Gamma_\psi}(\psi)\})^* - \Gamma^*_\psi) \\ &= ((\Gamma^* \cup \{\mu_{\Gamma_\psi}(\psi)\}) - \Gamma^*_\psi) \\ &= (\Gamma^* - \Gamma^*_\psi) \cup \{\mu_{\Gamma_\psi}(\psi)\}\end{aligned}$$

よって，$((\Gamma - \Gamma_\psi) \cup \{\mu_{\Gamma_\psi}(\psi)\})^* \subseteq (\Gamma^* - \Gamma^*_\psi) \cup \{\mu_{\Gamma_\psi}(\psi)\}$ より

$$((\Gamma - \Gamma_\psi) \cup \{\mu_{\Gamma_\psi}(\psi)\})^* = (\Gamma^* - \Gamma^*_\psi) \cup \{\mu_{\Gamma_\psi}(\psi)\}$$

(Q. E. D)

この制限は実質的には，プログラム Γ から ψ に関するサブプログラム Γ_ψ を隔離ものであり，$\langle not \rangle \psi$ があたかも事実として与えられているように扱うこと可能にすることを狙ったものである．このように考えればこの制限は $\langle not \rangle$ だけでなく，単調でも単射でもないヘッヂに対しても有効であるといえる．しかし，本論では $\langle not \rangle$ だけに関する考察にとどめる．

残る問題点は $\langle not \rangle$ が作用するリテラル ψ に許される解釈が唯一でなければならないという点であるが，これは命題 1によって解決される．

よって，$\langle not \rangle$ に対する制限 I，II の制限を与えられたファジィ論理プログラミングにおいては，健全性と完全性について議論できる．

定義 43 拡張導出規則

導出規則 I を適用する際，次の補助規則を適用することが出来る．

- *IV. negative hedge*
  規則式 $\phi \leftarrow \psi_1, \cdots, \langle h \rangle \psi_i, \cdots, \psi_n$ に関して

$$\mu_{\Gamma'}(\psi_1 \wedge \cdots \wedge \langle not \rangle \psi_i \wedge \cdots \wedge \psi_n) = \mu_\Gamma(\psi_1) \wedge \cdots \wedge (1 - \mu_\Gamma(\psi_i)) \wedge \cdots \wedge \mu_\Gamma(\psi_n)\}$$

補題 11 拡張導出規則 *IV.* は健全性を満たす．

Proof $/\psi_i/_T = Mi_\Gamma(\psi)$ であるような任意の妥当解釈 T において，定理 1, 定義 16, 式 (3.3), 命題 1より

$$
\begin{aligned}
&/\psi_1 \wedge \cdots \wedge \langle not \rangle \psi_i \wedge \cdots \wedge \psi_n/_T \\
&= /\psi_1/_T \wedge \cdots \wedge /\langle not \rangle \psi_i/_T \wedge \cdots \wedge /\psi_n/_T \\
&\geq \mu_\Gamma(\psi_1) \wedge \cdots \wedge (1 - \mu_\Gamma(\psi_i)) \wedge \cdots \wedge \mu_\Gamma(\psi_n) \\
&= \mu_{\Gamma'}(\psi_1 \wedge \cdots \wedge \langle not \rangle \psi_i \wedge \cdots \wedge \psi_n)
\end{aligned}
$$

よって，$Md^\Gamma(\psi_1 \wedge \cdots \wedge \langle not \rangle \psi_i \wedge \cdots \wedge \psi_n) \leq Mi^\Gamma(\psi_1 \wedge \cdots \wedge \langle not \rangle \psi_i \wedge \cdots \wedge \psi_n)$

(Q. E. D)

補題 12 拡張導出規則 $IV.$ は完全性を満たす．

Proof $\langle not \rangle$ の制限 II より，拡張導出規則 IV. は ψ_i のサブプログラムを除いたプログラムにおける解釈に制限を与えるものではない．

よって，拡張導出規則 IV. は完全性を満足する．

(Q. E. D)

5.4 拡張導出規則の完全性と健全性

既に定義されている導出規則に前節で新たに定義した言語ヘッヂに関する推論規則 (拡張導出規則 I, II, III, IV) を加えたものをまとめて拡張導出規則と呼ぶ．このとき，次のことがいえる．

定理 4 再帰的規則に対する制限，単調ヘッヂに対する制限，$\langle not \rangle$ に対する制限 I, II の 4 つの制限下において，言語ヘッヂをもつファジィ論理プログラミングの拡張導出規則は健全性を満たす．

Proof 定理 2，補題 7, 9, 11より明らか．

(Q. E. D)

定理 5 再帰的規則に対する制限，単調ヘッヂに対する制限，$\langle not \rangle$ に対する制限 I，II の 4 つの制限下において，言語ヘッヂをもつファジィ論理プログラミングの拡張導出規則は完全性を満たす．

Proof　定理 3，補題 8, 10, 12より明らか．

(Q. E. D)

以上の結果より言語ヘッヂを持つファジィ論理プログラミングの推論規則である拡張導出規則は，再帰的規則に対する制限，単調ヘッヂに対する制限，$\langle not \rangle$ に対する制限 I，II の 4 つの制限下において完全性と健全性を満たすことがいえた．

次の章では，これまで考察してきたファジィ論理プログラミングに基づくファジィ Prolog である LbFP を提案する．

第 6 章

Łukasiewiczの含意に基づくファジィ Prolog(LbFP)

ここでは，いままで考察してきたファジィ論理プログラミングに基づくファジィ Prolog である LbFP(Łukasiewicz's implication based Fuzzy Prolog) を提案する．

ここではまずプログラミング言語ファジィ Prolog を利用するユーザの立場になって，ファジィ Prolog がどのような機能を持つべきか，どのようなシステムであるべきかを考察し，純粋にファジィ論理プログラミングに基づく部分と，ファジィ論理プログラミングの範囲を超えているが必要と思われる部分に分けて LbFP システムを定義していく．

6.1 ユーザの要請分析

まず，ユーザの立場に立ってプログラミング言語であるファジィ Prolog への要請を分析する。これは，ファジィ Prolog システムは一体どのようなことに利用できるべきかを調べることに相当する．

仮定するユーザのレベルは，「プログラミング言語 Prolog をある程度知っていて，ファジィ理論を学んだことがあり，ファジィ Prolog は Prolog をファジィ化したものであるという漠然とした認識を持っている」という程度とする．つまりユーザの知識には Prolog に関する知識とファジィ理論に関する知識があるものとする．

Prolog の特徴的な機能、仕様は次のとおりである．

- 導出原理に基づく推論
- 単一化に基づくマッチング機能
- 推論のための探索機能

- 論理式（Horn 節）による宣言的なアルゴリズムの記述

実際，ユーザはこれらについて詳しい理論を知っているわけではなく，各々の特徴は次のようなイメージに投影されているといえる．

- Prolog は規則に基づいて書き換えを行うことにより動作する．
- 同じ形の記述があれば書き換えをする．変数はいかなる変数，定数とも等しいとみなすことができ，書き換えのときその変数をもう一方の変数，定数に書き換えることが出来る．
- 規則に基づいて可能な全ての書き換えを自動的に行う．
- 規則を使って物事の関係を記述する．問題を解くために必要な物事の関係を全て記述するだけでよい．

このイメージに基づいてユーザは Prolog を利用しているといえる． Prolog に対するイメージを総合すると「Prolog は規則に従って書き換えを自動的に，可能な限り行うもので，物事の関係をうまく捉えることによってプログラミングをする．」となるといえる．

ユーザにとってプログラミング言語 Prolog は，その他の手続き的アルゴリズムでは記述しづらいような処理をしたり，データベースのような非数値データを処理するための道具であり，数値処理を行ったり，手続き的な処理を行ったりする場合にわざわざ利用するものではない．

例えば， Prolog の例題として良く用いられる親子関係の問題や，手続き的には記述し難い n-Queen 問題のようなものに Prolog を利用することは，ごく自然であろう．しかし，クイックソートや 2 分木ソートのような手続き的アルゴリズムがはっきりしているようなものや，数値処理が苦手なのに数値計算に Prolog を使おうとするのは不自然である．

当然，本論で仮定するユーザも同様の傾向があるといえる．つまり， Prolog に対してユーザは数値処理や手続き的処理を求めてはいないとする．

ファジィ Prolog を「Prolog をファジィ化したもの」という漠然とした認識から捉えた場合，先ほど上げた Prolog の特徴から，ファジィ Prolog の特徴を次のように捉えることができるであろう．

- ファジィ導出原理に基づくファジィ線形導出推論
- ファジィ単一化に基づくファジィマッチング機能

- ファジィ推論のためのファジィ探索機能
- ファジィ論理式（Horn 節）による宣言的なファジィアルゴリズムの記述

ただし，個々の語の正確な意味は考慮する必要はない．重要なのは，これらの語からどのような機能がイメージされるかである．

- ファジィ Prolog はあい曖昧な規則に基づいて書き換えを行うことにより動作する．
- 似た形の記述があれば書き換えをする．変数はいかなる変数，定数とも等しいとみなすことができ，書き換えのときその変数をもう一方の変数，定数に書き換えることが出来る．
- 曖昧な規則に基づいて可能な全ての書き換えを自動的に行う．
- 規則を使って物事の曖昧な関係を記述する．問題を解くために必要な物事の関係をおおざっぱに記述するだけでよい．

これに基づいてファジィ Prolog に対するイメージを総合すると「ファジィ Prolog は曖昧な規則に従って書き換えを自動的に，可能な限り行うもので，物事の関係を曖昧に捉えることによってプログラミングをする．」となるといえる．

続いて，ユーザがファジィ理論の知識持っているという点からファジィ Prolog を捉えている．

ファジィ理論は，人間が扱うような曖昧な情報を体系的に扱う手段を与える理論であり，特に制御やエキスパートシステム，数値計画，データベース，認識，アルゴリズムなどの分野で大きな成果を挙げている．これらの分野に対してファジィ Prolog の利用が期待されると思われる．

ある研究 A にファジィ理論を適用して A^* が得られたとすると，これらの間には，普通 $A \subseteq A^*$ という関係が成り立つ．つまりファジィ理論を適用することは，研究がより一般的化されるということである．この点から考えると，Prolog のもとになる 2 値論理は，ファジィ論理に内包されているといえるので，当然，Prolog で出来ることはファジィ Prolog で出来なければならない．

つまり，ファジィ Prolog は Prolog の上位コンパチブルでなければならない．このことは，ファジィ Prolog の記述法が Prolog の記述法に準拠していなければならないことを意味する．

いま前述のとおり，ユーザは Prolog に対して数値処理や手続き的処理を求めてはいなかったので，ファジィ Prolog にも同様の立場を取るといえる．このことからファジィ Prolog をファジィ数値計画や手続き的ファジィアルゴリズムに利用することはないものと仮定することができる．またファジィ制御も応答関数をファジィ推論で近似するものであるので数値処理といえる．

以上のような点から，ユーザがファジィ Prolog に期待するあろう応用の分野は以下のようなものになるといえる．

- ファジィ推論
- ファジィエキスパートシステム
- ファジィデータベース
- 認識におけるファジィデータ比較
- 宣言的ファジィアルゴリズム

残念ながら本論で定義したファジィ論理プログラミングでは，これら全てを扱うことは出来ない．それは，1 階ファジィ述語では扱えないような処理が必要になるためである．

例えば，多段のファジィ推論においては度々用いられるファジィ集合の正規化は，タイプ II の言語ヘッヂに相当するし，ファジィエキスパートシステムでは「多くの (many)」のようなファジィ量限定子があるが，これは Σ カウントというファジィ測度を利用しているものであり，やはり本論のファジィ論理プログラミングでは扱えない．

Prolog にはプログラムに新たな知識を加える機能やカットオペレータ，数式処理など，論理プログラムの範囲ではないような機能が含まれており，重要な役割を果たしている．そこで LbFP にもファジィ論理プログラミングでは議論できないが必要と思われるような機能を導入することとする．

まずは簡単に一般的 Prolog システムを紹介し，純粋にファジィ論理プログラミングに基づく部分，次に付加機能について示すこととする．

6.2 Prolog システム

Prolog システムは 2 値論理プログラミングに基づく論理型プログラミング言語である．

プログラムは事実と規則と呼ばれる 2 種類の書式の集合として記述される．それぞれが論理プログラミングにおける事実節，規則節を表現している．

例 6		論理プログラミング	*Prolog*
	事実	$honest(tom)$	`honest(tom).`
	規則	$\forall(x)(decent(x) \leftarrow honest(x))$	`decent(X) :- honest(X).`

大文字の X は変数を意味している．これにより式中の変数が明らかであるので全称束縛子を記述する必要はない．

述語の項には定数項と変数項がある．これらの区別は項を表す文字列の 1 文字目で行われる．

定数項 アルファベット小文字，もしくは数字から始まる文字列

変数項 アルファベット大文字，もしくはアンダースコア (_) から始まる文字列

さらに構造を持った項として関数項とリスト項がある．

関数項 func(T1,T2,···,Tn) のような関数形式の項

リスト項 [T1,T2,···,Tn] のような順番を持った項の集合

もちろん T1 などに関数項やリスト項を取ることも出来る．

関数項の関数名に述語と同じものを用いることも可能であるが，混乱のもとになるので避けるべきである．

例 7 Prolog プログラムの例

「*tom* は正直 *(honest)* である.」「*tom* は礼儀正しい *(polite).*」「*bob* は礼儀正しい.」という *3* つの事実と，「正直で礼儀正しい人は, 社会的に信頼される *(decent).*」という規則を論理プログラミングで考えると, そのプログラムは

$$\{honest(tom),\ polite(tom),\ polite(bob),\ \forall(x)(decent(x) \leftarrow honest(x), polite(x))\}$$

となる. これを *Prolog* で記述すると次のようになる。

```
honest(tom).
polite(tom).
polite(bob).

decent(X):-honest(X),polite(X).
```

Prolog で処理を実行するためには問い合わせ (質問) と呼ばれる入力を行う．この入力が推論のトリガとなって反駁法により推論が行われる．Prolog は既に存在しているプログラムに勝手に推論規則を適用して無秩序に推論結果を生成して行くというようなものではない．

Prologシステムは基本的に問い合わせ形式以外の入力は行われず，プログラムなどは組み込み述語を問い合わせに用いることにより，システムに組み込まれる．一般に問い合わせは，

```
?-decent(X).
```

のように入力される．?-decent(X). は $\forall(x)(0 \leftarrow decent(x))$ を意味する．

問い合わせた内容がプログラムの推論結果となる場合は，推論が成功したことをユーザに知らせ，もし先ほどの例のように問い合わせ中に変数が含まれていて推論課程でその変数が代入によって具体化されている場合，同時に変数にどのような代入がされたかを知らせる．さらにそれ以外の代入が可能かどうかを問い合わせることも出来る．

例 8 Prologの推論動作例例 7のプログラムに対して，問い合わせ?-decent(X). をすると，X に tomがマッチして，decent(tom) が得られるが,bob は honest(bob). という事実が存在しないので，decent(bob) とはならない．

6.3 LbFP システム I - 純粋ファジィ論理プログラミング部 -

LbFP システムは Prolog システムの上位コンパチブルでなければならない．よって，

- プログラムは規則と事実の集まりとして構成され，問い合わせを行うことにより推論動作を実行する．
- 述語の取る項には，定数項と変数項の他，関数項という構造を持った項や順番を持った項の列のリスト項が許される．

といった基本的部分は Porlog と同じである．

本節では Prolog システムには含まれない，ファジィ論理プログラミングに基づく LbFP 固有の表記法，機能について述べ， Prolog と全く同じ部分については割愛する．

なお，正確な文法は付録に BNF 法を用いて掲載してある．

6.3.1 再帰的規則に対する制限

前の章で述べたとおり，本論のファジィ論理プログラミングにおいては，推論過程で再帰的に同じリテラルが現われることが禁止されている．しかし，実際の LbFP システムにおいてプログラムが組み込まれた時点でこれをチェックすることは現実的とはいえない．

なぜなら，プログラムから推論可能なすべての推論結果を得てからでないとチェックできないためである．そこで，問い合わせにより推論が実行された時点で，再帰的に同じリテラルが現われるかどうかをチェックするという方法を取ることとする．

また，この制限に反することで必ず導出規則の健全性が破綻するという訳ではないので，再帰的に同じリテラルが現われる場合でも，システムはユーザに対して警告を行うにとどめるものとする．

6.3.2 真理値

LbFP がもっとも Prolog と異なる点は，個々の事実や規則に真理値が与えられる点である．ファジィ論理プログラミングのプログラム Γ において $\mu_\Gamma(honest(tom)) = 0.9$ であるとき，LbFP では `honest(tom)[0.9].` として真理値に [] を付けた形で記述する．

$\mu_\Gamma(decent(x) \leftarrow honest(x)) = 1$ のように真理値が 1 の場合は省略し，`decent(x) :- honest(x).` となる．

例 9 *LbFP* プログラムの例
例 *4* を *LbFP* で記述すると次のようになる．

```
decent(X) :- honest(X).
decent(X) :- polite(X).
honest(tom)[0.9].
polite(tom)[0.8].
honest(bob)[0.9].
polite(bob)[0.85].
```

この真理値を用いることにより，ファジィ集合を表現することが可能であることはいうまでもない．ただし，項同士の関係 (例えば，数値の順序関係) は考慮されないので，台集合は必ず離散的に扱われる．

例 10 ファジィ集合の表現
*kid(*子供*)* という語でラベルづけされるようなファジィ集合

$$\{1/0, \cdots, 1/10, 0.9/11, 0.8/12, 0.7/13, 0.5/14, 0.3/15, 0.2/16, 0.1/17\}$$

を *LbFP* で表現すると，

```
young(0).
```

vdots

```
young(10).
young(11)[0.9].
young(12)[0.8].
young(13)[0.7].
young(14)[0.5].
young(15)[0.3].
young(16)[0.2].
young(17)[0.1].
```

のようになる.

6.3.3 言語ヘッヂ

言語ヘッヂは次の6つの基本ヘッヂを採用する.

- `very` 非常に ($\langle very \rangle$)
- `morl` ほぼ ($\langle morl \rangle$)
- `plus` 強調化 ($\langle plus \rangle$)
- `minus` 弛緩化 ($\langle minus \rangle$)
- `exact` 厳密に ($\langle exact \rangle$)
- `not` でない ($\langle not \rangle$)

前章で論じたとおり，指数型ヘッヂである `very`， `morl`， `plus`， `minus` は任意の述語に作用することができ，単調ヘッヂの制限，$\langle not \rangle$ の制限Ｉより `exact`， `not` は規則の本体においてのみ述語に作用できる.

$\langle very \rangle decent(x)$ は `very * decent(X)` と記述し，$\langle plus \rangle \langle plus \rangle decent(x)$ は `plus * plus * decer` と記述するものとする.

特に $\langle not \rangle$ については作用する述語に関するサブプログラムが分離可能でなければならないが，システムレベルでのチェックはプログラムが組み込まれたり修正されるたびにそれを行う必要があり，これに反するプログラムに対しては警告を行う必要がある．

6.4 LbFP システム II - 非ファジィ論理プログラミング部 -

本節では，Prolog におけるカットオペレータのようにファジィ論理プログラミングの範囲では扱えないが，実用的システムとするために必要な機能について述べる．前節同様 LbFP は基本的に Prolog に準拠するので，カットオペレータやその他の組み込み述語の機能はそのまま引き継ぐものとして，ここでは，LbFP 固有の機能について述べる．

ユーザの要請分析より，LbFP には次の分野への利用が予想される．

- ファジィ推論
- ファジィエキスパートシステム
- ファジィデータベース
- 認識におけるファジィデータ比較
- 宣言的ファジィアルゴリズム

これらの問題を解決する上で必要な手法で，ファジィ論理プログラミングでは扱うことが出来ないものとして

- ファジィ集合の正規化
- レベル n ファジィ集合
- ファジィ項 (ファジィ数)
- ファジィ集合の基数 (Σ カウント)
- ファジィ量限定 (many など)

が挙げられる．

ファジィ集合の直積や射影は規則を記述することにより可能であるし，ファジィ質限定は言語ヘッヂを用いることにより可能である．（ただし，指数型ヘッヂに限られる．）

ここではこれらについて考察し，LbFP システムに導入出来るかどうかを判断し，導入できるものについてはその扱いについて述べる．

6.4.1 正規化

ファジィ集合 P の正規化ファジィ集合 $NORM(P)$ は P のメンバシップ関数を μ_P とすると次のように定義される．

$$\mu_{NORM(P)}(x) = \nu \times \mu_P$$

ただし $\times$ は算術積であり，$\nu = \sup_{\forall x}\{\mu_P(x)\}$ である．

正規化そのものはスカラー倍の言語ヘッヂとして定義できるが，ν は 1 階ファジィ述語論理では得られない．これが正規化がファジィ論理プログラミングの範囲ではない理由である．しかし，ν を既知の値と考えれば正規化を単調ヘッヂと同様に扱うことが可能であることがいえる．また，正規なファジィ集合に正規化を行っても恒等ヘッヂと同様に効果はない．

LbFP においてファジィ集合 P を p(X) で扱うとすると，正規化を行うためには，p(X) が取る最大の真理値を知る必要がある．LbFP で実現するためには X に代入可能な全ての要素に対して推論を行い，それらの結果の最大値を nu とする．推論において再び p(X) が用いられる可能性もあるので，得られた結果はプログラムが書き換えられるまでシステムが保持しておくと，それ以降の動作の効率化を行うことが出来る．

動作が言語ヘッヂとほぼ同じであることから，norm * p(X) というように言語ヘッヂと同じ記述を行うものとする．

6.4.2 レベル n ファジィ集合

レベル n ファジィ集合は，台集合が $[0,1]$ であるようなファジィ集合で表わされるファジィ真理値を真理値として扱う必要がある．

ファジィ真理値を取るファジィ論理は一般的には Kleene 代数のモデルとはならない．これが，レベル n ファジィ集合がファジィ論理プログラミングの範囲ではない理由である．ファジィ真理

値の導入を認めるためにはファジィ真理値を取るファジィ論理プログラミングを考察しなければならない．また，ファジィ真理値と普通の真理値が混在すると混乱をきたす恐れもある．

これらの結果から LbFP システムではレベル n ファジィ集合は扱わないこととする．

補足：レベル n ファジィ集合は述語の次元を n-1 増やすことにより，普通の真理値を取るファジィ集合に書き換えることが出来るので，その手法を用いることで LbFP システムでレベル n ファジィ集合を扱うことも可能である．ただし，述語の次元を n-1 増やすことによって記述量が膨大になることが予想されるので現実的な解決法とはいえない．

6.4.3 ファジィ項

ファジィ項はファジィ数のように 1 つの対象をファジィ集合で扱うもので，ファジィ論理プログラミングにおいては述語の項にファジィ集合で特徴づけられるような項を認めるというものである．関数項の関数をメンバシップ関数として解釈すれば実現可能であるといえる．

ただし項同士の単一化を行う場合に，ファジィ集合同士のマッチングをどのように定義し，結果として得られるマッチング結果やマッチングの度合などをファジィ論理プログラミングでどのように扱うべきであるかは十分議論する必要がある．さらにファジィ集合をファジィ述語として扱うことにしている LbFP では，関数項もファジィ集合を表わすことになると述語と関数の区別がなくなってしまう．

正規化は高々 2 階のファジィ述語であるが，ファジィ項の導入はそれ以上の高階ファジィ述語につながる．

これらの結果から LbFP システムではファジィ項は扱わないこととする．

6.4.4 ファジィ集合の基数

ファジィ集合の基数はファジィ集合に含まれる要素の個数を数え上げるためのものであり，実際は 2 つのファジィ集合の基数の比率から真理値を求めるもので，ファジィ量限定と共に用いられる．さらに要素の数をファジィ数であらわすものもあるが，ファジィ項のところでファジィ数を扱うことを否定したので，シングルトンの実数値で基数を扱う．なお，このときえられる結果は真理値ではなく正の実数である．

ファジィ集合の基数は Σ カウントという手法で与えられる．いま，ファジィ集合 P ののメンバシップ関数を μ_P とすると Σ カウント $\Sigma Count(P)$ は次のように定義される．

$$\Sigma Count(P) = \Sigma_{\forall x} \mu_P(x)$$

ただし Σ は算術総和を求めるもので，ファジィ集合を定義する論理和ではない．

このままの結果では $\Sigma Count(P)$ は真理値ではなく正の実数である． Σ カウントを用いて真理値を得るためには相対 Σ カウントという手法を用いる．

これは，2つのファジィ集合の基数の比率を求めるもので，例えば A を「独身」 B を「若い」というラベルで特徴退けられたファジィ集合とすると，相対 Σ カウント $\Sigma Count(A/B)$ は「若ければ独身である」という度合を表わすものである．

相対 Σ カウントの定義は

$$\Sigma Count(A/B) = \frac{\Sigma Count(A \cap B)}{\Sigma Count(B)}$$

である．

ファジィ論理プログラミングの規則が個々の項に対しての真理値を与えるのに対して，相対 Σ カウントはファジィ集合同士の関係に対して真理値を与えるものであるが， $\Sigma Count(A/B)$ は明らかに $A \leftarrow B$ の真理値を意味するものである．

そこで， LbFP では `p(X)/q(X)` というように記述することとし，相対 Σ カウントの持つ意味は「若ければ独身である」のように規則と考えられるので，規則 `p(X) :- q(X).` の真理値として `p(X) :- q(X)[p(X)/q(X)].` のように記述することとする．ただし `p(X) :- q(X)[p(X)/q(X)].` のように記述上は規則であるが，実際には事実 `p(X)[p(X)/q(X)].` として扱われる．

また，規則で出てくる述語 `p`，`q` と真理値で出てくる述語 `p`，`q` は同一であってもなくても良い．

6.4.5 ファジィ量限定

ファジィ量限定を行なうファジィ量限定子は，相対 Σ カウントで得られた真理値に作用する関数として定義される．先ほどの例において相対 Σ カウント $\Sigma Count(A/B)$ で得られる結果は「若ければ独身である」の度合を表わしていたが，詳しくは「若い」集合に属しているなら「独身」集合に属しているという頻度を意味する．

ファジィ量限定子「ほとんど」を例に考える．「ほとんど」という語は「大部分が正しければ正しい」という意味に解釈することができる．すると例えば，結果が1であれば『「ほとんど」の「若い」集合に属している要素は「独身」集合に属している』ことが度合1で正しいと解釈でき，0.7程度であれば，厳密には0.7程度しか正しくはないが「ほとんど」という修飾詞がついているので0.8程度正しいというように多少甘めに解釈することができる．

この「ほとんど」がファジィ量限定子であり，頻度を修飾する修飾詞である．ファジィ量限定は次のグラフような1変数関数として与えられるもので，数値的には言語ヘッヂと同様の効果をもたらす．

LbFPシステムでは，ファジィ量限定子として「全て」，「ほとんど」，「多く」とその否定的表現「少し」，「ほとんどない」，「全くない」を基本的なファジィ量限子定として与える．

all　全て　$\begin{cases} 1 & (\text{if } T(\chi) = 1) \\ 0 & (\text{otherwise}) \end{cases}$

most　ほとんど

many　多く

little　少し

few　ほとんどない

never　全くない

また，相対Σカウントに対して作用する言語ヘッヂの様なものであるので，`p(X) :- q(X)[all * p(X)/q(X)].` のように記述するものとする．文法上は相対Σカウントでなく，具体的な値で与えられている真理値に適用してもよい．(全く意味はなさないが)

また，ファジィ述語のための言語ヘッヂもファジィ量限定した真理値に作用できるものとする．例えば `p(X) :- q(X)[very * many * p(X)/q(X)].` のようにでき(「very many」という表現は言葉としては正しくないかもしれないが)，これにより非常に多くのファジィ量限定を作ることが出来る．ただし，言語ヘッヂのみで相対Σカウントに作用することは出来ないものとする．

6.4.6 その他のLbFP固有の機能

LbFPでは，Prologの機能に加え，

- 真理値の割り当て
- 言語ヘッヂ
- 正規化
- 相対Σカウントによる真理値
- 真理値のファジィ量限定

を，固有の機能として認めたが，ここでもう 1 つ，ファジィ推論に LbFP を用いるために必要な機能を加える．

LbFP でファジィ推論を行うためには，ファジィ推論の前提が事実としてプログラム中に存在している必要がある．しかし実際にファジィ推論を行う場合，前提は推論を実行するたびに変更されるものである．前提を変更するためには，前提の記述部分をシステム上から削除し，別の前提の記述を組み入れる直す必要がある．

この入れ替えを効率よく行うために，プログラムの事実を 2 種類に分けることにする．

恒久事実 システム動作の途中で変更されることにない事実．

一時事実 システム動作の途中で頻繁に変更される事実で，削除，入れ替えのための専用の組み込み述語 (`tempReset`) により，簡単に入れ替えが出来る．

一時事実の記述には，記述の先頭に % をつけて，`%honest(tom)[0.9].` のようにする．ただし，恒久事実か一時事実かはユーザの責任で決めるものであり，全ての事実を恒久事実としても一時事実としても全く問題はない．また，恒久事実でも，プログラムデータを操作する組み込み述語を利用することによって入れ替えは可能である．

LbFP は以上のような機能をもったファジィ論理プログラミング言語として実装された．次の章ではユーザの要請分析で得られた分野に実際に LbFP システムを利用する例をいくつか示す．

第 7 章

LbFP を用いたファジィ問題の解決

本章では計算上に実装された LbFP システムを用いて，実際の問題に適用してその有用性を示す．

本章で用いた LbFP システムは，Sun Microsystems 社の OS である SunOS4.1.3 において開発されたものであり，現段階 (1996 年 2 月 10 日現在) では正規化とファジィ量限定子はサポートしていない．

プログラミング言語には C を用い，構文解析部を yacc で，字句解析部を lex で記述してある．LbFP システムのソースコードは付録 D として掲載してある．

7.1 ファジィ推論

ここでは，ファジィ推論への適用の例として，簡単なファジィ推論の例と岩石の種類を分類するファジィ推論システムを示す．

7.1.1 簡単なファジィ推論

まずは，簡単な例として「たくさん食べると太る」というファジィ推論を考える．

IF 「食べる量が多い」 **THEN** 「体重が増える」

これを LbFP で実現したものが，次のプログラムである．

```
/* Premises */
%4timesEat(3)[0.5].
%4timesEat(4).
%4timesEat(5)[0.5].
```

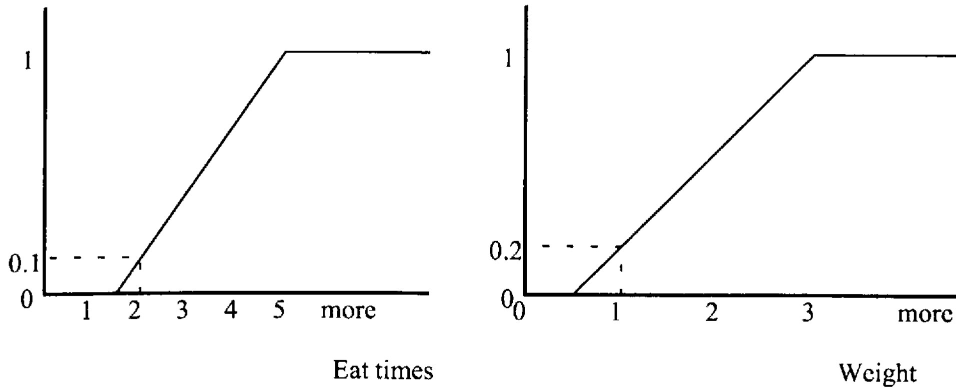

図 7.1: 食べる量と体重の変化

```
/* Matching */
eat(X):-4timesEat(X).

/* Rules */
fat(0):-eat(2)[0.9].
fat(1):-eat(2).
fat(2):-eat(2).

⋮

fat(3):-eat(5)[0.6].
fat(more):-eat(more).
```

なお，/*...*/ はコメントである．

LbFP システムは数値も他の文字列と同様に扱われるので，ファジィ集合は離散的に与えられなければならず，また入力されるデータに対してのメンバシップ値が提供されていなければならない．ここでは入力されるデータから必要な点を調べ，メンバシップ値を手作業で抽出した．将来的には連続的に与えたメンバシップ関数から自動的にルールを生成するようにすべてきであろう．

このプログラムは前提がファジィ集合で与えられており，前提の変更が頻繁に行われる可能性が高いので一時事実として %4timesEat(4). のようにしてある.

事実とファジィ推論の前提をマッチングさせるためにコメント /*Matching*/ で示される部分がある. 前提ファジィ集合をあらわすファジィ述語に，異なった名前(この場合は eat と tt 4timesEat) を付けなければならないのでこの様な記述となる.

問い合わせは?-fat(X). で行われ，結果は各 X ごとの真理値が得られるので，それをファジィ集合として解釈する. 結果としては，

```
?-fat(X).
X=0
[0.3];
X=1
[0.5];
X=2
[0.9];
X=3;
X=more
```

となり，言語的に解釈すれば，「まあまあ太る」となる.

7.1.2 岩石分類システム

岩石分類システムは，[23] で構築されたシステムである.

岩石分類システムの目的は，火成岩系の岩石を判別することであり，ボーリングによって得られたサンプルから専門家以外でも客観的に取得可能な，火成砕屑度, 平均粒径, 色指数, 斑晶量 , 角レキの有無，という 4 種類のデータからその岩石の種類を 19 種類のうちのいずれであるかを判別する. 判別のための知識 (ルール) は専門家によって与えられたものである.

その知識を元に，次の様なファジィ推論のための知識が生成された. (一部抜粋)

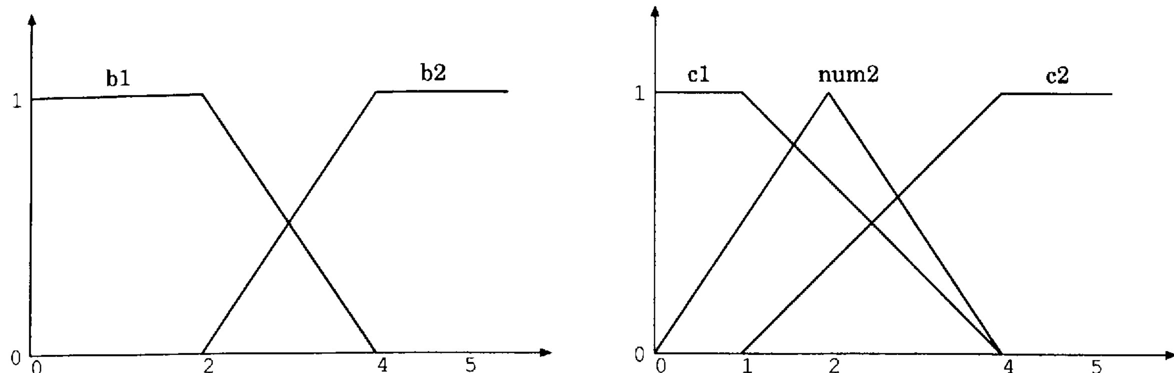

図 7.2: 火成砕屑度と平均粒径

IF 「火成砕屑度が B1」 ∧ 「平均粒径が C1」 ∧ 「色指数が F1」 ∧ 「斑晶量が H1」
THEN 「岩石は緻密流紋岩」
IF 「火成砕屑度が B1」 ∧ 「平均粒径が C1」 ∧ 「色指数が F1」 ∧ 「斑晶量が H2」 ∧ 「角レキが I1」
THEN 「岩石は流紋岩」
IF 「火成砕屑度が B1」 ∧ 「平均粒径が C1」 ∧ 「色指数が F1」 ∧ 「斑晶量が H3」
THEN 「岩石は斑状流紋岩」
IF 「火成砕屑度が B2」 ∧ 「平均粒径が C2」 ∧ 「平均粒径が D1」 ∧ 「色指数が F2」
THEN 「岩石は凝灰角レキ岩」
IF 「火成砕屑度が B1」 ∧ 「平均粒径が C1」 ∧ 「色指数が F1」 ∧ 「斑晶量が H2」 ∧ 「角レキが I2」
THEN 「岩石は角レキ流紋岩」
IF 「火成砕屑度が B1」 ∧ 「平均粒径が C3」 ∧ 「色指数が F4」 ∧ 「斑晶量が H3」
THEN 「岩石はドレライト」

B1 は火成砕屑度の定義域上に定義されたファジィ集合であり，B2,C1 などもそれぞれファジィ集合を表わす．これらの集合には特にラベルづけは行われていないが，「大きい」などのラベルづけも可能である．

火成砕屑度と平均粒径で定義されているファジィ集合は次のとおりである．

このシステムを LbFP で記述したものが付録 C である．以下にその一部を示す．

```
/* Fuzzy Sets on B */
b1(2)[1.0].
b1(2.2)[1.0].
b1(1.5)[1.0].
```

```
b1(1)[1.0].

/* Fuzzy Sets on C */
c1(0.2)[1.0].
c1(0.3)[1.0].
c1(0.5)[1.0].
c1(0.1)[1.0].
c1(0.4)[1.0].
c1(0.8)[1.0].
c1(0.6)[1.0].
c1(1)[1.0].
c1(2)[0.0].
c1(1.8)[0.0].
c1(1.5)[0.0].

/* Rules */
rule(s1,b(B),c(C),d(D),f(F),h(H),i(I)):-b1(B),c1(C),f1(F),h1(H).
rule(s2,b(B),c(C),d(D),f(F),h(H),i(I)):-b1(B),c1(C),f1(F),h2(H),no(I).
rule(s3,b(B),c(C),d(D),f(F),h(H),i(I)):-b1(B),c1(C),f1(F),h3(H).
rule(s13,b(B),c(C),d(D),f(F),h(H),i(I)):-b2(B),c2(C),d1(D),f2(F).
rule(s18,b(B),c(C),d(D),f(F),h(H),i(I)):-b1(B),c1(C),f1(F),h2(H),yes(I).
rule(s19,b(B),c(C),d(D),f(F),h(H),i(I)):-b1(B),num2(C),num40(F),over40(H).

/* Query */
result(X,B,C,D,F,H,I):-rule(X,b(B),c(C),d(D),f(F),h(H),i(I)).
```

岩石分類システムは入力が必ずシングルトンなので，前の例のように前提を事実としてプログラムせずに，問い合わせのときに与えるようにしてある．

動作例は，付録 C に掲載してある．

7.2 宣言的ファジィアルゴリズム

ここでは，宣言的ファジィアルゴリズムの例として簡単な宣言的ファジィアルゴリズムの例とファジィオートマトンを扱う．

7.2.1 簡単な宣言的ファジィアルゴリズム

ここでは，Prolog でよく知られている親子関係を友達関係にアレンジしたものを示す．

親子関係のプログラムとは「親の先祖は先祖である」というもので，次のようなプログラムで記述される．

```
parent(pam,bob).
parent(tom,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).
parent(pat,jim).

predecessor(X,Y):-parent(X,Y).
predecessor(X,Z):-parent(X,Y),predecessor(Y,Z).
```

述語 parent(X,Y) は「X は Y の親である」という述語である．

このとき?-predecessor(tom,jim). と問い合わせれば yes. というような解答が得られる．

これを友達関係にアレンジして見ると，つぎのようなプログラムが得られる．ここで重要なのは，友達関係は親子関係と比べると曖昧であり，「友達の友達は，だいたい友達である」という関係が成り立つということである．

```
friend(pam,bob).
friend(bob,pam).
friend(tom,pam).
friend(pam,tom).
friend(liz,tom).
friend(tom,liz).
```

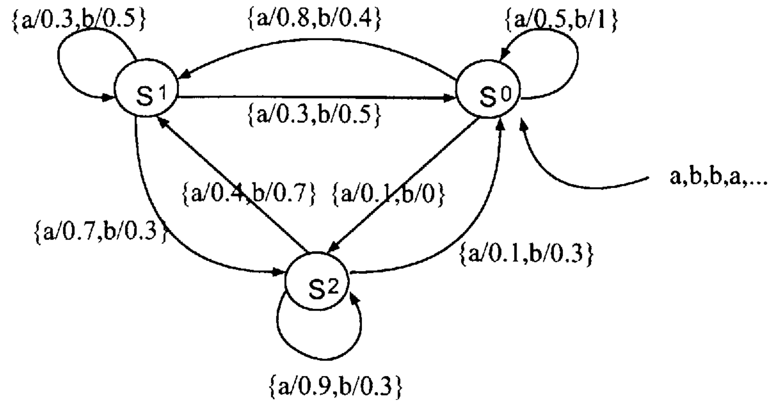

図 7.3: ファジィオートマトン

```
friend(X,Z):-friend(X,Y),friend(Y,Z),not * eq(X,Z)[0.8].
```

前提の友達関係は対称的であり，一方的に友達だと思っているようなことはないものとした．また簡単のために，友達関係には真理値が与えていない．

規則において，eq(X,Z) というものがあるが，これは「X と Z は等しい」という組み込み述語である．ただし，eq はファジィ述語ではない．この not * eq(X,Z) で，規則が同一の友達関係に繰り返し利用されるのを防いでいる．

問い合わせ?-friend(liz,pam). は，0.8 で成り立つことが分かる．また，?-friend(bob,X). とすれば，bob の友達を調べることができ，結果として X=pam; X=tom[0.8]; X=liz[0.6] が得られる．

7.2.2 ファジィオートマトン

ファジィオートマトンとは，状態の遷移に度合をもつような非決定性オートマトンのことである．図 7.3において，例えば状態 S_0 からの遷移は，入力語が "1"のとき S_0 へは 0.2，S_1 へは 0.9，S_2 へは 0.5 の度合で遷移することを意味している．度合は遷移確率ではないことに注意しなければならない．

ファジィオートマトンでは，各状態１つ１つが台集合の要素となっている．そして各状態の度合がメンバシップ値となって，システム全体で１つのファジィ集合 (概念) を表現するというもの

である.

ここでは，例として病気の状態と薬の投与に関するファジィオートマトンシステムを示す.

7.2.3 病気の状態と薬の投与に関するファジィオートマトン

今, ある人の状態が「病が重い」であるとする. そして, その人が病を治すために薬を飲んでいった時のその人の病状の変化について考えることにする. 症状は, 状態 S_1 から S_3 の 3 つの要素からなる台集合上のファジィ集合として表現されるものとする. S_1 が「脈が正常である」, S_2 が「だるい」, S_3 が「体温が高い」という要素であるとする.

入力としては, 薬 a,b の 2 種類を考える. 各薬について

a : かなりよく効くが副作用も大きい

b : 効果は小さめだが, 副作用はほとんどない

というような性質を持っているとする.

例えば状態 S_2 である時に薬 a を服用すると, 状態 S_1 への遷移の度合は 0.9 と高いが, 状態 S_3 への遷移の度合も 0.7 と高めである. 症状としてはまた, 薬 b を服用すると状態 S_1 への遷移の度合は 0.7 でさほど高くはないが, 状態 S_3 への遷移の度合は 0.1 とかなり低く, これは病状はめざましく良くはならないが病状がさらに悪化する事は殆どない事を意味している.

すべての状態と入力における遷移を定義すると状態遷移図は図 7.4のようになる.

ここで今, ある人の病状がかなり重い場合, 例えば状態 S_1,S_2,S_3 である度合をそれぞれ 0.1,0.3,1 で表される場合を考える. まず, 薬 b よりも薬 a をたくさん服用した場合, その度合を $\mu_U(a) = 1$,$\mu_U(b) = 0.3$ とする. この時, 次の状態は $\mu_X(x_t + 1)$ は次の式で求められる.

$$\mu_X(x_{t+1}) = \bigvee x_t \bigvee u_t[\mu_X(x_t) \wedge \mu_U(u_t) \wedge \mu_X(x_{t+1} \mid x_t, u_t)]$$

よって, 次の状態では,S_1,S_2,S_3 である度合はそれぞれ 0.9,0.7,0.7 となる.

次に, 薬 b を多く服用した場合, その度合を $\mu_U(a) = 0.3$,$\mu_U(b) = 1$ とした時, 上の式より, 次の状態では,S_1,S_2,S_3 である度合はそれぞれ 0.3,0.7,0.6 となる.

プログラムと動作例は付録 C に掲載してある.

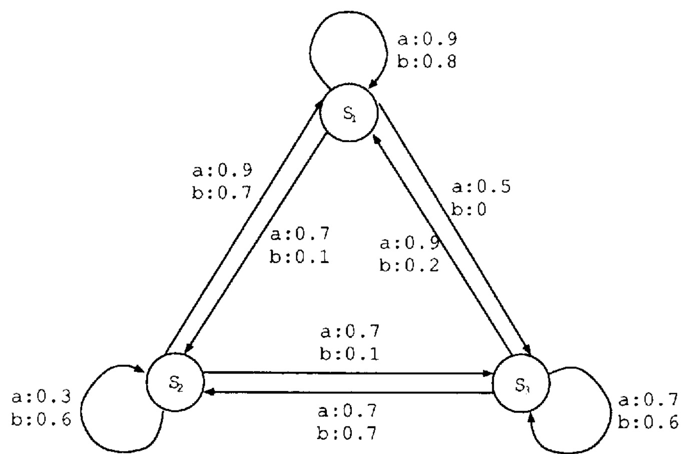

図 7.4: 病気の状態と薬の投与に関するファジィオートマトン

第 8 章

結論

本論文では，まずファジィ論理プログラミングにおいて，推論規則が満たすべき 2 つの性質を要請という形で示し，その要請に基づいて推論規則 modus ponens における含意演算と連言演算に関する考察を行った．その結果として

- Łukasiewicz の含意 - Łukasiewicz の積
- Gödel の含意 - 論理積
- Goguen の含意 - 代数積

が，最も適した組み合わせであることを示した．さらに，Łukasiewicz 含意とŁukasiewicz 積の持つ特徴から，Łukasiewicz 含意 - Łukasiewicz 積がこれらの中で最もファジィ論理プログラミングに向いていることを示した．

さらに，Lukasiewicz の含意を用いたファジィ論理プログラミングを定義し，推論規則が完全性と健全性を満足することを示した．

続いて，ファジィ論理プログラミングで扱うことの出来る対象を広げるために，ファジィ述語の修飾詞である言語ヘッジを導入し，その推論規則を考察した．その結果として

- ファジィ論理プログラミングにおける再帰的規則に対する制限
- 単調ヘッジは規則本体にしか適用できないという制限
- $\langle not \rangle$ は規則本体にしか適用できないという制限
- $\langle not \rangle$ の適用された述語におけるサブプログラムに対する制限

という制限の下で，言語ヘッヂを導入したLukasiewicz の含意を用いたファジィ論理プログラミングにおける推論規則が完全性と健全性を満足することを示した．

本論文で導入したLukaisewicz の含意と1項論理演算である言語ヘッヂをファジィ論理に導入した場合の，代数的構造などの考察は今後の課題となる．

本論文ではさらに，提案した言語ヘッヂを導入したLukasiewicz の含意を用いたファジィ論理プログラミングに基づいて，ファジィ Prolog システムである LbFP(Łukaisewicz's implication based Fuzzy Prolog) の仕様を検討し，ファジィ論理プログラミングの範囲では扱えないが，ファジィ問題を扱う上で重要な正規化と相対 Σ カウント，ファジィ量限定の導入を決め，ファジィ問題を解決するためのプログラムをより簡単に記述できるようにした．

最後に LbFP システムを用いて，ファジィ問題を解決する例をいくつか示し， LbFP がファジィ問題を扱うプログラミング言語として有用なものであることを示すことが出来た．

今後は LbFP システムを公開し，さらにさまざまなファジィ問題に適用して，現在の機能の確認やさらに必要な機能について検討する必要がある．またファジィ集合などは, テキスト形式で入力するのはかなり手間がかかるので, GUI なども導入してより有用な LbFP システムを構築することが必要である．

謝辞

Łukasiewicz の含意を用いたファジィ論理プログラミングの研究に対して, 多大な御指導・御鞭撻を頂きました向殿政男教授に厚く御礼を申し上げます。先生の御助力がなければ本論文の完成はなかったでしょう.

格別のご配慮により, 快適な研究環境を与えて下さいました中所武司教授に深謝申し上げます.

そして, 何より多くの御助言・御協力を頂きました明治大学理工学部情報科学科の先生方, 博士後期課程への進学に際しましてお世話になりました電子通信工学科遠藤哲郎教授に深謝申し上げます.

研究生活の多岐に渡り, 多大な御助力を頂きました粕谷慎一氏を始めとする情報科学科システム科学研究室およびソフトウェア工学研究室の皆様方に, 改めに謝意を申し上げます. そして, 最後に大変お世話になりました明治大学理工学事務室の方々に御礼申し上げます.

研究活動の素晴らしさを論理数学の深さを教えて下さいました故駒宮安男先生とその家族の方々に深謝申し上げます. 先生にお会いしなければ今の私はなかったはずです.

以上の方々の御協力がなければ本研究の完成は不可能でした.

最後になりましたが, 長期に渡る学生生活を快く認めて下さり, 研究生活を影から支えて下さいました両親に改めて心よりの深謝申し上げます. 本当にありがとうございました.

参考文献

[1] Dubois D., Lang J., Prade H.: Fuzzy sets in approximate reasoning, Part1: Inference with possibility distributions. Fuzzy Sets and Systems 40(1991)143-202

[2] Dubois D., Lang J., Prade H.: Fuzzy sets in approximate reasoning, Part2: Logical approaches. Fuzzy Sets and Systems 40(1991)203-244

[3] Dubois D., Prade H.: A theorem on implication functions definded from triangular norm. Stochastica 8(3)(1994)267-279

[4] Van Emden M.: Quantitative Deduction and its Fixpoint Theory. The journal of Logic Programming 4, 1(1986)37-53

[5] Hindel C.J.: Fuzzy Prolog. International Journal of Man-Machine Studies 24(1986)569-595

[6] Ishizuka M., Kanai N.:Prolog-Elf incorporating fuzzy logic. Proc. of the 9th International Joint Conference on Artificial Intelligence(IJCAI 85), Los Angels(1985)701-703

[7] Ivánek J., Švenda J., Ferjenčik J./: Inference in expert systems based on complete multivalued logic. Kybernetika, Proceeding of the Workshop on Uncertainty Processing in Expert Systems, Alšovice, Czechoslovakia(1988)25-32

[8] Kikuchi H., Mukaidono M.: PROFIL: Fuzzy Interval Logic Prolog. Prep. of the International workshop on Fuzzy Systems Applications(IFSA), Iizuka(1988)205-206

[9] 菊池 浩明, 向殿 政男: ファジィ論理プログラムの線形導出 日本ファジィ学会誌, Vol. 6 No. 2(1994)294-303

[10] 井関 清志: 記号論理学 (命題論理) 槙書店 (1973)

[11] 井関 清志: 記号論理学 (述語論理) 槙書店 (1973)

[12] Klawonn F., Kruse R.: A Lukasiewicz Logic Based Prolog. , Mathware & Soft Computing 1(1994)5-29

[13] Kowalski R.: Predicate Logic as a Programming Language. , Proc. of IFIP Congress. North-Holland, Amsterdam(1974)569-574

[14] Kowalski R.: Algorithm = Logic + Control., Comm. ACM 22(1979)424-436

[15] Lee R.C.T., Chang C.L.: Some properties of fuzzy logic. Information and Control 19(1971)417-431

[16] Lee R.C.T., : Fuzzy logic and the resolution principle. Journal of the Assoc. for Computing Machinery 19(1972)109-119

[17] LeFaivre R.A.: The presentation of fuzzy knowledge. Journal of Cybernetics 4, 2(1974)57-66

[18] Martin T.P., Baldwin J.F., Pilsworth B.W.: The implementation of FPROLOG - A fuzzy Prolog interpreter. Fuzzy Sets and Systems 23(1987)119-129

[19] Mukaidono M.: Fuzzy inference in resolution style. In: Fuzzy Sets and Possibility Theory - Recent Developments. Edited by R. R. Yager, Pergamon Press(1982)224-231

[20] 長尾真, 淵一博 論理と意味, 岩波講座 情報科学 - 7

[21] 長尾真 知識と推論, 岩波講座 ソフトウェア科学 -14(1988)

[22] Robinson J.A.: A Machine Oriented Logic Based on the Resolution Principle. Journal of ACM 12(1)(1965)23-41

[23] 李 瑞平, 向殿 政男: ファジィクラスタリングと岩石分類への応用 システム・制御研究会 (電気学会)(1994)

[24] Shen Z.L., Ding L., Mukaidono M.: A theoretical framework of fuzzy Prolog machine. In: Fuzzy Computing - Theory, Hardware and Applications(M. M. Gupta, T. Yamakawa, eds.). North-Holland, Amsterdam(1988)89-100

[25] Sterling L.,Shapiro E.:The Art of Prolog(2nd edition). MIT Press, Cambridge MA(1986)

[26] Umano M.:Fuzzy-Set Prolog. Preprints of 2nd IFSA Congress, Tokyo(1987)750-753

[27] Yager R.R.: Inference in a multivalued logic system. International Journal of Man-Machine Studies 23(1985)27-44

[28] Yasui H., Mukaidono M.: Postulates and proposals on Fuzzy Prolog. 2nd European Congress on Intelligent Techniques and Soft Computing(EUFIT '94), Aachen vol. 2(1994)1080-1086

[29] Yasui H., Hamada Y., Mukaidono M.:Fuzzy Prolog Based on Łukasiewicz Implication and Bounded Product. International Joint Conference of the 4th IEEE International Conference on Fuzzy Systems and the 2nd International Fuzzy Engineering Symposium(FUZZ-IEEE/IFES95), vol. III (1995)949-955

[30] 安井 浩之, 向殿 政男: Łukasiewicz の含意を用いたファジィ論理プログラムに関する研究 日本ファジィ学会誌

[31] Zadeh L.A.: Fuzzy Sets. Information and Control 8(1965)338-353

[32] Zadeh L.A.: A Fuzzy-Set-Theoretic Interpretation of Linguistic Hedges. Journal of Cybernetics 2(1972)4-34

[33] ZadehL.A.: PRUF — A Meaning Representation Language for Natural Languages. Int. J. Man-Machine Studies 10(1978)395-460

[34] 菅野道夫, 向殿政男監訳: ザデー・ファジィ理論 (Fuzzy Set and Applications-Selected Papers by L.A. Zadeh 1987). 日刊工業新聞社 (1992)

付録 A

証明

ただし，$*$ は任意の指数型ヘッヂを，$+$ は任意の強調化ヘッヂを，$-$ は任意の弛緩化ヘッヂを，$\langle h^c \rangle$ は単射ヘッヂ $\langle h \rangle$ の補ヘッヂを各々表わすのものとする．

(11) $a \leftarrow b = \neg b \leftarrow \neg a$

Łukaiewicz の含意の定義 $(a \leftarrow b = \min\{a - b + 1, 1\})$ より，

$$a \leftarrow b = \min\{a - b + 1, 1\} = \min\{(1 - b) - (1 - a) + 1, 1\} = \neg b \leftarrow \neg a$$

(12) $a \leftarrow a = 1$

Łukaiewicz の含意の定義 $(a \leftarrow b = \min\{a - b + 1, 1\})$ より，

$$a \leftarrow a = \min\{a - a + 1, 1\} = 1$$

(13) $(a \leftarrow b) \leftarrow c = (a \leftarrow c) \leftarrow b$

Łukaiewicz の含意の定義 $(a \leftarrow b = \min\{a - b + 1, 1\})$ より，

$$(a \leftarrow b) \leftarrow c = \min\{\min\{a - b + 1, 1\} - c + 1, 1\}$$

$$= \min\{a - b - c + 2, 2 - c, 1\} = \min\{a - b - c + 2, 1\}$$

$$(a \leftarrow c) \leftarrow b = \min\{\min\{a - c + 1, 1\} - b + 1, 1\}$$

$$= \min\{a - c - b + 2, 2 - b, 1\} = \min\{a - b - c + 2, 1\}$$

(14) $(a \leftarrow b) \leftarrow a = 1$

Łukaiewicz の含意の定義 $(a \leftarrow b = \min\{a - b + 1, 1\})$ より,

$$(a \leftarrow b) \leftarrow a = \min\{\min\{a - b + 1, 1\} - a + 1, 1\}$$

$$= \min\{a - b - a + 2, 2 - a, 1\} = \min\{2 - b, 2 - a, 1\} = 1$$

(15) $*_1 *_2 a = *_2 *_1 a$

指数ヘッヂの定義 $(\langle * \rangle a = (a)^{\gamma}, \gamma > 0)$ より,

$$*_1 *_2 a = ((a)^{\gamma_2})^{\gamma_1} = (a)^{\gamma_2 \gamma_1} = (a)^{\gamma_1 \gamma_2} = ((a)^{\gamma_1})^{\gamma_2} = *_2 *_1 a$$

(16) $*(a \wedge b) = *a \wedge *b,\ *(a \vee b) = *a \vee *b$

指数ヘッヂの定義 $(\langle * \rangle a = (a)^{\gamma}, \gamma > 0)$ より,

$$*(a \wedge b) = (\min\{a, b\})^{\gamma} = \min\{(a)^{\gamma}, (b)^{\gamma}\} = *a \wedge *b$$

同様に, $*(a \vee b) = *a \vee *b$

(16') $\langle exact \rangle (a \wedge b) = \langle exact \rangle a \wedge \langle exact \rangle b,\ \langle exact \rangle (a \vee b) = \langle exact \rangle a \vee \langle exact \rangle b$

$\langle exact \rangle$ の定義 $(\langle exact \rangle a = 1 (a = 1), 0 (a < 1))$ より,

- if $a = b = 1$

  $a \wedge b = 1$ より,

  $$\langle exact \rangle (a \wedge b) = 1 = \langle exact \rangle a \wedge \langle exact \rangle b$$

- otherwise

  $a \wedge b < 1$ であり, 少なくとも a, b のいずれか一方が 1 でないので,

  $$\langle exact \rangle (a \wedge b) = 0 = \langle exact \rangle a \wedge \langle exact \rangle b$$

同様に，$\langle exact\rangle(a \vee b) = \langle exact\rangle a \vee \langle exact\rangle b$

(17) $-a \leftarrow a = 1$

$-a \geq a$ より，明らか.

(18) $a \leftarrow +a = 1$

$a \geq +a$ より，明らか.

(18') $a \leftarrow \langle exact\rangle a = 1$

$a \geq \langle exact\rangle a$ より，明らか.

(19) $\langle very\rangle a = \langle plus\rangle\langle plus\rangle\langle plus\rangle a$

定義より，

$$\langle plus\rangle\langle plus\rangle\langle plus\rangle a = (a)^{\sqrt[3]{2}\sqrt[3]{2}\sqrt[3]{2}} = (a)^2 = \langle very\rangle a$$

(20) $\langle h\rangle\langle h^c\rangle a = a$

補ヘッヂの定義より明らか.

付録 B

LbFP 文法

LbFP の構文を BNF 表記で以下に示す．なお表記中「()」で囲まれているものは非終端記号を表わしており，「''」で囲まれているものは終端記号を表わすものとする．

(program)	:	(program) (expression)
	|	(expression)
(query)	|	'?' bodies '.'
(expression)	:	'.'
	|	(head) '.'
	|	(head) (truth) '.'
	|	'%' (head) '.'
	|	'%' (head) truth '.'
	|	(head) ':' '-' (bodies) '.'
	|	(head) ':' '-' (bodies) (truth) '.'
(head)	:	(hPredicate)
(bodies)	:	(bodies) ',' (bPredicate)
	|	(predicate)

```
(hPredicate)  :  (hHedges) '*' (pName) '(' (terms) ')'
              |  (hHedges) '*' (pName)
              |  (pName) '(' (terms) ')'
              |  (pName)
(bPredicate)  :  (bHedges) '*' (pName) '(' (terms) ')'
              |  (bHedges) '*' (pName)
              |  (pName) '(' (terms) ')'
              |  (pName)
(hHedges)     :  (hHedge) '-' (hHedges)
              |  (hHedge)
(bHedges))    :  (bHedge) '-' (bHedges)
              |  (hHedge) '-' (bHedges)
              |  (bHedge)
(hHedge)      :  'v' 'e' 'r' 'y'
              |  'm' 'o' 'r' 'l'
              |  'p' 'l' 'u' 's'
              |  'm' 'i' 'n' 'u' 's'
(bHedge)      :  'n' 'o' 't'
              |  'e' 'x' 'a' 'c' 't'
```

(terms) : (terms) ',' (term)
| (term)
(term) : (list)
| (tName) '(' (terms) ')'
| (tName)
| (variable)
| (number)
(list) : '[' (terms) ']'
| '[' (term) '|' (variable) ']'
| '[' (term) '|' (list) ']'
| '[' ']'
(truth) : '[' (tDigits) ']'
| '[' (term)'/' (term) ']'
| '['(hedge) '*' (predicate)'/' (predicate) ']'
(pName) : (sLetter) (string)
| (sLetter)
(tName) : (sLetter) (string)
| (sLetter)
(variable) : (cLetter) (string)
| '_' (string)
| (sLetter)
| '_'
(string) : (string) (letter)
| (letter)
(letter) : (sLetter)
| (cLetter)
| (digit)

```
(number)    :  (uDigits) '.' (digits)
            |  (uDigits)
(tDigits)   :  '0' '.' (digits)
(uDigits)   :  (uDigit) (digits)
            |  (uDigit)
(digits)    :  (digits) (digit)
            |  (digit)
(digit)     :  '0'|(uDigit)
(sLetter)   :  'a'|'b'|'c'|'d'|'e'|'f'|'g'|'h'|'i'|'j'|'k'|'l'|'m'|'n'|'o'
            |  'p'|'q'|'r'|'s'|'t'|'u'|'v'|'w'|'x'|'y'|'z'
(cLetter)   :  'A'|'B'|'C'|'D'|'E'|'F'|'G'|'H'|'I'|'J'|'K'|'L'|'M'|'N'|'O'
            |  'P'|'Q'|'R'|'S'|'T'|'U'|'V'|'W'|'X'|'Y'|'Z'
(uDigit)    :  '1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'
```

付録 C

LbFP プログラムと動作例

C.1 岩石分類システム

C.1.1 ソースコード

```
b1(2)[1.0].
b1(2.2)[1.0].
b1(1.5)[1.0].
b1(1)[1.0].

c1(0.2)[1.0].
c1(0.3)[1.0].
c1(0.5)[1.0].
c1(0.1)[1.0].
c1(0.4)[1.0].
c1(0.8)[1.0].
c1(0.6)[1.0].
c1(1)[1.0].
c1(2)[0.0].
c1(1.8)[0.0].
c1(1.5)[0.0].
num2(0.2)[0.0].
num2(0.3)[0.0].
num2(0.5)[0.0].
num2(0.1)[0.0].
num2(0.4)[0.0].
num2(0.8)[0.0].
num2(0.6)[0.0].
num2(1)[0.3].
num2(2)[1.0].
num2(1.8)[1.0].
num2(1.5)[1.0].

f1(12)[1.0].
f1(10)[1.0].
f1(13)[1.0].
f1(15)[1.0].
```

```
f1(14)[1.0].
f1(40)[0.0].
f1(42)[0.0].
f1(38)[0.0].
f1(43)[0.0].
f1(45)[0.0].
num40(12)[0.0].
num40(10)[0.0].
num40(13)[0.0].
num40(15)[0.0].
num40(14)[0.0].
num40(40)[1.0].
num40(42)[1.0].
num40(38)[1.0].
num40(43)[1.0].
num40(45)[1.0].

h1(7)[1.0].
h1(10)[1.0].
h1(12)[1.0].
h1(15)[0.4].
h1(5)[1.0].
h1(20)[0.0].
h1(30)[0.0].
h1(35)[0.0].
h1(25)[0.0].
h1(42)[0.0].
h1(50)[0.0].
h1(48)[0.0].
h1(40)[0.0].
h1(45)[0.0].
h1(55)[0.0].
h1(60)[0.0].
h2(7)[0.0].
h2(10)[0.0].
h2(12)[0.0].
h2(15)[0.6].
h2(5)[0.0].
h2(20)[1.0].
h2(30)[1.0].
h2(35)[0.6].
h2(25)[1.0].
h2(42)[0.0].
h2(50)[0.0].
h2(48)[0.0].
h2(40)[0.0].
h2(45)[0.0].
h2(55)[0.0].
h2(60)[0.0].
h3(7)[0.0].
h3(10)[0.0].
h3(12)[0.0].
h3(15)[0.0].
h3(5)[0.0].
h3(20)[0.0].
```

```
h3(30)[0.0].
h3(35)[0.4].
h3(25)[0.0].
h3(42)[1.0].
h3(50)[1.0].
h3(48)[1.0].
h3(40)[1.0].
h3(45)[1.0].
h3(55)[0.0].
h3(60)[0.0].
over40(7)[0.0].
over40(10)[0.0].
over40(12)[0.0].
over40(15)[0.0].
over40(5)[0.0].
over40(20)[0.0].
over40(30)[0.0].
over40(35)[0.0].
over40(25)[0.0].
over40(42)[0.0].
over40(50)[0.5].
over40(48)[0.0].
over40(40)[0.9].
over40(45)[0.4].
over40(55)[1.0].
over40(60)[1.0].

no(0)[0.7].
no(10)[1.0].

rule(s1,b(B),c(C),d(D),f(F),h(H),i(I)):-b1(B),c1(C),f1(F),h1(H).
rule(s2,b(B),c(C),d(D),f(F),h(H),i(I)):-b1(B),c1(C),f1(F),h2(H),no(I).
rule(s3,b(B),c(C),d(D),f(F),h(H),i(I)):-b1(B),c1(C),f1(F),h3(H).
rule(s19,b(B),c(C),d(D),f(F),h(H),i(I)):-b1(B),num2(C),num40(F),over40(H).

result(X,B,C,D,F,H,I):-rule(X,b(B),c(C),d(D),f(F),h(H),i(I)).
```

C.1.2 動作例

本システムは，地層から客観的に得られる，火成砕屑度 (B), 平均粒径 (C) (D), 色指数 (F), 斑晶量 (H), 角レキ (I) の各データを，質問?-result(X,B,C,D,F,H,H). を介して入力し，その岩石の種類を判断するというものである.

動作結果を以下に示す.

```
Welcome to FProlog/LbFP ver. 1.5.1

?result(X,2,0.3,0.3,12,10,20).

X = s1

[1.000000]
```

```
;
No.

?-result(X,2,0.4,0.4,13,15,25).
X = s1
[0.020000]
;
X = s2
[0.990000]
;
No.

?-result(X,2,0.8,0.8,15,40,25).
X = s3
[1.000000]
;
No.
```

1つめの問い合わせの答えは，岩石がs1(緻密流紋岩)であることを意味している．noの表示はこれ以上答えがないことを意味する．

2つめの問い合わせの答えはs1である度合が0.02でs2(流紋岩)である度合が0.99であることを意味している．

各々の答えは，元々の岩石分類システムと同じ答えであり，専門科の判断と一致している．

C.2 ファジィオートマトン

C.2.1 ソースコード

```
trans(ux(x1),ui(a),ux(x1))[0.9].
trans(ux(x1),ui(a),ux(x2))[0.7].
trans(ux(x1),ui(a),ux(x3))[0.5].
trans(ux(x2),ui(a),ux(x1))[0.9].
trans(ux(x2),ui(a),ux(x2))[0.3].
trans(ux(x2),ui(a),ux(x3))[0.7].
trans(ux(x3),ui(a),ux(x1))[0.9].
```

```
trans(ux(x3),ui(a),ux(x2))[0.7].
trans(ux(x3),ui(a),ux(x3))[0.7].

trans(ux(x1),ui(b),ux(x1))[0.8].
trans(ux(x1),ui(b),ux(x2))[0.1].
trans(ux(x1),ui(b),ux(x3))[0].
trans(ux(x2),ui(b),ux(x1))[0.7].
trans(ux(x2),ui(b),ux(x2))[0.6].
trans(ux(x2),ui(b),ux(x3))[0.1].
trans(ux(x3),ui(b),ux(x1))[0.2].
trans(ux(x3),ui(b),ux(x2))[0.7].
trans(ux(x3),ui(b),ux(x3))[0.6].


ux(x1)[0.1].
ux(x2)[0.3].
ux(x3)[1].

ui(a)[0.3].
ui(b)[1].

final(ux(x1)).
final(ux(x2)).
final(ux(x3)).

result(ui([Y]),ux(FI)):-ux(X),ui(IN),u(ux(X),ui(Y),ux(FI)).
result(ui([CAR|CDR]),ux(FI)):-result(CAR),ui(CDR),u(result(CAR),ui(CDR),ux(FI)).


result(ux(FI)):-ux(X),ui(l(Z)),u(ux(X),ui(l(Z)),ux(FI)).

accept(ux(S),[X]):-
trans(ux(S),ui(X),ux(S1)),ux(S),ui(X),final(ux(S1)).
```

C.2.2 動作例

```
Welcome to FProlog/LbFP ver. 1.5.1

?-result(X,[1]).

X = x1

[0.300000]

;

X = x2

[0.700000]

;

X = x3

[0.600000]
```

```
;
no.
?-result(X,[2]).
X = x1
[0.900000]
;
X = x2
[0.700000]
;
X = x3
[0.700000]
.
no.
?-result(X,[2,1]).
X = x1
[0.800000]
;
X = x2
[0.700000]
;
X = x3
[0.600000]
.
```

まず`?-result(X,[1]).`は,1 のパターンの投与を意味し,0.3/ui(1,a),1/ui(1,b), つまり薬 a を 0.3, 薬 b を 1 の度合で投与した場合である. この A は変数であり, 問い合わせの際は大文字であれば何でも良い. そして, 問い合わせをした結果, 変数 A に度合が代入され結果として返ってくる. この場合は, 患者が薬を投与された後の状態は,$x1$ である度合が 0.3,$x2$ である度合が 0.7,$x3$ である度合が 0.6 である事が分かる.

さらに`?-result(X,[2,1]).`のような問い合わせも可能である. これは, 薬の投与の 2 のパターンを行なった後でさらに, 1 のパターンを投与するという意味である. このように入力はリスト形

式であるため, 個数に制限されることはない.

この結果はファジィオートマトンの定義式により得られる結果と等しいものとなっている. このようにFuzzy Prologを用いてFuzzy Automatonのようなファジィシステムの記述を行なえることが示せた.

付録 D

LbFP システム ソースコード

LbFP システム ソースコード

D.1 メインと構文解析部

D.1.1 fpro.yacc

```
%{
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>;
#include <sys/wait.h>;
#include <math.h>
#include <signal.h>

#include"version.h"
#include"fpro.h"

%}

%union{
Term* term;
Stack* stack;
Name* name;
double num;
Hedge h;
Hedge* hedges;
}

%token <name> VAR NAME NUM

%start start
```

```
%%

/* start is start token */
/* clause forms such as followings:
certainty/head:-body1,body2,..,bodyn.[truth]
?certainty/body1,body2,..,bodyn.[truth]
certainty/ and/or [truth] can be omitted. (default value 1.0 is used)
*/
start : /* empty */ {
if( yyin == stdin ){
unput('?');
fprintf(yyout,"?");
}
}
| start clause {
if( yyin == stdin ){
unput('?');
fprintf(yyout,"?");
}
}
;
clause : '.'  {
}
| head '.'  {
recordClause($<term>1, NULL, 1.0, 1.0);
}
| head truth '.'  {
recordClause($<term>1, NULL, 1.0, $<num>2);
}
| certainty head '.'  {
recordClause($<term>2, NULL, $<num>1, 1.0);
}
| certainty head truth '.'  {
recordClause($<term>2, NULL, $<num>1, $<num>3);
}
| '%' head '.'  {
recordTmpClause($<term>2, NULL, 1.0, 1.0);
}
| '%' head truth '.'  {
recordTmpClause($<term>2, NULL, 1.0, $<num>3);
}
| '%' certainty head '.'  {
recordTmpClause($<term>3, NULL, $<num>2, 1.0);
}
| '%' certainty head truth '.'  {
recordTmpClause($<term>3, NULL, $<num>2, $<num>4);
}
| head ':' '-' bodies '.'  {
recordClause($<term>1, $<stack>4, 1.0, 1.0);
}
| head ':' '-' bodies truth '.'  {
recordClause($<term>1, $<stack>4, 1.0, $<num>5);
}
| certainty head ':' '-' bodies '.'  {
```

```
recordClause($<term>2, $<stack>5, $<num>1, 1.0);
}
| certainty head ':' '-' bodies truth '.'  {
recordClause($<term>2, $<stack>5, $<num>1, $<num>6);
}
| '?' bodies '.'  {
resolve($<stack>2, 1.0, 1.0);
}
/* | '?' bodies truth '.'  {
resolve($<stack>2, 1.0, $<num>3);
}*/
| '?' certainty bodies '.'  {
resolve($<stack>3, $<num>2, 1.0);
}
/* | '?' certainty bodies truth '.'  {
resolve($<stack>3, $<num>2, $<num>4);
}*/
| error '.'{ yyerror(": invalid clause\n"); fprintf(yyout,"%s\n",yytext); yyerrok; yyclea
}
;
/* head is Pred*(Term*) */
head : predicate {
$<term>$ = $<term>1;
}
;

/* bodies is Stack* */
bodies : bodies ',' predicate {
$<stack>$ = addPredicates($<stack>1, stackPredicate($<term>3));
}
| predicate {
$<stack>$ = stackPredicate($<term>1);
}
;

/* predicate is Pred*(Term*) */
predicate: hedges '*' NAME '(' terms ')' {
$<term>$ = recordPredicate($<hedges>1, $<name>3, $<stack>5);
}
|hedges '*' NAME {
$<term>$ = recordPredicate($<hedges>1, $<name>3, NULL);
}
|NAME '(' terms ')' {
$<term>$ = recordPredicate(NULL, $<name>1, $<stack>3);
}
| NAME {
$<term>$ = recordPredicate(NULL, $<name>1, NULL);
}
;

/* hedges is Hedge* */
hedges: hedge '-' hedges {
$<hedges>$ = addHedge($<h>1, $<hedges>3);
}
| hedge {
```

```
$<hedges>$ = addHedge($<h>1, NULL);
}
;
/* hedge is Hedge* */
hedge: NAME  {
$<h>$ = encodeHedge($<name>1, NULL);
}
;

/* terms is Stack* */
terms : terms ',' term {
$<stack>$ = addTerms($<stack>1,stackTerm($<term>3));
}
| term {
$<stack>$ = stackTerm($<term>1);
}
;

/* term is Term*(Var*) */
term : list {
$<term>$ = $<term>1;
}
| NAME '(' terms ')' {
$<term>$ = recordFunction($<name>1, $<stack>3);
}
| NAME {
$<term>$ = recordTerm($<name>1);
}
| VAR {
$<term>$ = recordVariable($<name>1);
}
| NUM {
$<term>$ = recordTerm($<name>1);
}
;

/* list is Term* */
list : '[' terms ']' {
$<term>$ = list($<stack>2);
}
| '[' term '|' VAR ']' {
$<term>$ = listCarCdr($<term>2, recordVariable($<name>4));
}
| '[' term '|' list ']' {
$<term>$ = listCarCdr($<term>2, $<term>4);
}
| '[' ']' {
$<term>$ = emptyList;
}
;

/* truth & certainty */
truth : '[' digit ']' {
$<num>$ = $<num>2;
}
```

```
| '[' VAR ']' {
$<num>$ = (double) VARIABLE * varID(recordVariable($<name>2));
}
;

certainty : digit '/' {
$<num>$ = $<num>1;
}
| VAR '/' {
$<num>$ = (double) VARIABLE * varID(recordVariable($<name>2));
}
;

/* digit is double */
/* digit is a value which is included in the closed interval [0,1] */
digit : NUM {
$<num>$ = inter01($<name>1);
}
;
%%

#include "lex.yy.c"

/* procedure for output */
void foutput(file,string,mode)
FILE *file;
char *string;
char *mode;
{
char view_mode;
if(mode){
switch(*mode){
case ('m'):
case ('M'):
view_mode = (char)1;
break;
case ('d'):
case ('D'):
view_mode = (char)2;
break;
case ('c'):
case ('C'):
view_mode = (char)3;
break;
case ('t'):
case ('T'):
default:
view_mode = (char)4;
break;
}
} else {
view_mode = (char)4;
}
fwrite(file,&view_mode,1,1);
fwrite(file,string,1,len);
```

```
fwrite(file,
}

/* procedures for allocation */

/* ????* alloc?????() allocate ???? type date */
Name* allocName()
{
Name* name;
name = (Name*) malloc(sizeof(Name));
name->string = NULL;
name->flag = NONVAR;
return(name);
}

Term* allocTerm()
{
Term* term;
term = (Term*) malloc(sizeof(Term));
term->hedge = NULL;
term->name = NULL;
term->terms = NULL;
term->size = 0;
return(term);
}

#define allocVar allocTerm
#define allocPred allocTerm

Clause* allocClause()
{
Clause* clause;
clause = (Clause*) malloc(sizeof(Clause));
clause->head = NULL;
clause->bodies = NULL;
clause->size = 0;
clause->truth = 0.0;
clause->certainty = 0.0;
return(clause);
}

Substitute* allocSubstitute()
{
Substitute* subs;
subs = (Substitute*) malloc(sizeof(Substitute));
subs->var = NULL;
subs->target = NULL;
subs->next = NULL;
return(subs);
}

Stack* allocStack()
{
Stack* stack;
stack = (Stack*) malloc(sizeof(Stack));
```

```
stack->this = NULL;
stack->next = NULL;
return(stack);
}

IStack* allocIStack()
{
IStack* istack;
istack = (IStack*) malloc(sizeof(IStack));
istack->this = 0;
istack->truth = 0.0;
istack->bodyTruth = 0.0;
istack->hedge = NULL;
istack->pre = NULL;
return(istack);
}

CStack* allocCStack()
{
CStack* cstack;
cstack = (CStack*) malloc(sizeof(CStack));
cstack->this = NULL;
cstack->substitute = NULL;
cstack->index = 0;
cstack->istack = NULL;
cstack->truth = 0.0;
cstack->certainty = 0.0;
cstack->pre = NULL;
return(cstack);
}

/* Term** allocTerms(int, Stack*) allocates area and convert stack to array */
Term** allocTerms(size, stack)
Stack* stack;
int size;
{
Term** terms;
Stack* index;
int i;
void checkRedundant();
if(!size) return(NULL);
if(!(terms = (Term**) malloc(sizeof(Term*) * size))){
fprintf(stderr,"Terms allocation is failed");
return(NULL);
}
for(index = stack, i = 0; index; index = index->next, i++)
terms[i] = index->this;
/*freeStack(stack);*/
checkRedundant(terms,size);
return(terms);
}

/* procedures for initialzation */

/* void initDB(void) initializes each data bases */
```

```
void initDB()
{
/*void builtIn();*/

termDB.DBSize = 0;
predDB.DBSize = 0;
varDB.DBSize = 0;
tmpTermDB.DBSize = 0;
tmpTermDB.TmpVarIndex = 0;
clauseDB.DBSize = 0;
clauseDB.DBIndex = 0;
tmpClauseDB.DBSize = 0;
tmpClauseDB.DBIndex = 0;
}

/* void initConst() initializes some constant */
void initConst()
{
static char listString[] = "?list";
static Clause* eClause;
listName = allocName();
listName->string = listString;
listName->flag = NONVAR;
emptyList = allocTerm();
emptyList->name = listName;
emptyList->size = 0;
emptyList->terms = NULL;
termDB.elem[termDB.DBSize++] = emptyList;
eClause = allocClause();
eClause->size = 0;
eClause->head = NULL;
eClause->bodies = NULL;
emptyStack = allocCStack();
emptyStack->this = eClause;
}

/* procedures for comparation */

/* int roughTermcmp(Term*, Term*) rough-tests whether both are same or not */
/* if same then return 0 */
int roughTermcmp(term1, term2)
Term* term1;
Term* term2;
{
int i;
if(term1->name->flag || term2->name->flag) return(0);
if(term1->size != term2->size) return(1);
if(namecmp(term1->name, term2->name)) return(1);
for(i = 0; i < term1->size; i++)
if(roughTermcmp(term1->terms[i], term2->terms[i])) return(1);
return(0);
}

/* int termcmp(Term*, Term*) tests whether both are same or not */
/* if same then return 0 */
```

```
int termcmp(term1, term2)
Term* term1;
Term* term2;
{
int i;
if(term1->size != term2->size) return(1);
if(namecmp(term1->name, term2->name)) return(1);
for(i = 0; i < term1->size; i++)
if(termcmp(term1->terms[i], term2->terms[i])) return(1);
return(0);
}

/* int namecmp(Name*, Name*) tests whether both are same or not */
/* if same then return 0 */
int namecmp(name1, name2)
Name* name1;
Name* name2;
{
if(name1->flag != name2->flag) return(1);
if(!name1->string && !name2->string) return(0);
return(strcmp(name1->string, name2->string));
}

/* int cmpAryStk(Term**, int, Stack*) tests terms and stack are equivalence */
/* if same then return 0 */
int cmpAryStk(terms, size, stack)
Term** terms;
int size;
Stack* stack;
{
int i;
Stack* index;

if(size != depth(stack)) return(1);
for(i = 0, index = stack; index; i++, index = index->next)
if(termcmp(terms[i], index->this)) return(1);
return(0);
}

/* procedures for testing */

/* int emptyClause(CStack*) tests whether clause is empty clause or not */
int emptyClause(target)
CStack* target;
{
return(!target->this->size);
}

/*int includesTmpVar(Term*) tests whether arguments include tmpVar or not */
/* if include then return 1 */
int includesTmpVar(term)
Term* term;
{
int i;
if(term->name->flag == TMPVAR) return(1);
```

```
for(i = 0; i < term->size; i++)
if(includesTmpVar(term->terms[i])) return(1);
return(0);
}

/* procedures for printing */

/* void print???(????) prints context */
void printList(term)
Term* term;
{
void printTerm();
if(term->size){
printTerm(term->terms[0]);
if(namecmp(term->terms[1]->name, listName)){
fprintf(yyout,"| ");
printTerm(term->terms[1]);
} else if(term->terms[1]->size){
fprintf(yyout,", ");
printList(term->terms[1]);
}
}
}

void printTerm(term)
Term* term;
{
int i;
void printHedge();

if(!namecmp(term->name, listName)){
fprintf(yyout,"[");
printList(term);
fprintf(yyout,"]");
} else {
printHedge(term->hedge);
fprintf(yyout,"%s", term->name->string);
if(term->size){
fprintf(yyout,"(");
for(i = 0; i < term->size - 1; i++){
printTerm(term->terms[i]);
fprintf(yyout,", ");
};
printTerm(term->terms[term->size - 1]);
fprintf(yyout,")");
}
}
}

void printHedge(hedge)
Hedge* hedge;
{
int i;
if(!hedge) return;
/* if(hedge[0] == NORMAL) return;*/
```

```
for(i = strlen(hedge); i > 0; i--){
switch(hedge[i - 1]){
case(NOT):
fprintf(yyout,"not");
break;
case(VERY):
fprintf(yyout,"very");
break;
case(MORL):
fprintf(yyout,"morl");
break;
case(PLUS):
fprintf(yyout,"plus");
break;
case(MINUS):
fprintf(yyout,"minus");
break;
case(EXACT):
fprintf(yyout,"exact");
break;
default:
fprintf(yyout,"normal");
break;
}
fprintf(yyout,"%s",(i==1)?"*":"-");
}
}

void printClause(clause)
Clause* clause;
{
int i;

if(clause->certainty != 1.0)
fprintf(yyout,"%8.6f/",clause->certainty);
if(clause->head){
printTerm(clause->head);
if(clause->size) fprintf(yyout,":");
} else{
fprintf(yyout,"?");
}
if(clause->size){
fprintf(yyout,"-");
for(i = 0; i < clause->size - 1; i++){
printTerm(clause->bodies[i]);
fprintf(yyout,", ");
}
printTerm(clause->bodies[clause->size - 1]);
}
if(clause->truth != 1.0)
fprintf(yyout,"[%8.6f]",clause->truth);
fprintf(yyout,".\n");
}

/* void printSubstitute(Substitute*) prints contexts of substitutes */
```

```
void printSubstitute(subs)
Substitute* subs;
{
Substitute* index;
for(index = subs; index; index = index->next){
if(index->var){
#ifndef FPRODEBUG
if(index->var->name->flag != TMPVAR){
if(!includesTmpVar(index->target)){
#endif
printTerm(index->var);
fprintf(yyout," = ");
printTerm(index->target);
fprintf(yyout,"\n");
#ifndef FPRODEBUG
}
}
#endif
}
}
}

/* procedures for searching */

/* Term* searchTermDB(Name*, Stack*) search a term which has same name and arity*/
Term* searchTermDB(name, terms)
Name* name;
Stack* terms;
{
int i;
for(i = 0; i < termDB.DBSize; i++)
if(!namecmp(name,termDB.elem[i]->name)){
/* free(name->string);
free(name);
name = termDB.elem[i]->name;*/
if(cmpAryStk(termDB.elem[i]->terms, termDB.elem[i]->size, terms)) return(NULL);
return(termDB.elem[i]);
};
return(NULL);
}

/* Var* searchVarDB(Name*) search whether varDB includes Name* */
Var* searchVarDB(name)
Name* name;
{
int i;
for(i = 0; i < varDB.DBSize; i++)
if(!namecmp(name,varDB.elem[i]->name)){
/* free(name->string);
free(name);
name = varDB.elem[i]->name;*/
return(varDB.elem[i]);
};
return(NULL);
}
```

```
/* Pred* searchPredDB(Hedge*, Name*, Stack*) search a predicate which has same name and
arity */
Pred* searchPredDB(hedge, name, terms)
Hedge* hedge;
Name* name;
Stack* terms;
{
int i;
if(!hedge) hedge = "=";
for(i = 0; i < predDB.DBSize; i++)
if(!namecmp(name,predDB.elem[i]->name) && !strcmp(hedge,predDB.elem[i]->hedge)) {
if(cmpAryStk(predDB.elem[i]->terms, predDB.elem[i]->size,terms)) return(NULL);
return(predDB.elem[i]);
}
return(NULL);
}

/* int searchClauseDB(Pred*, Stack*) search a clause which has same head and bodies on
ClauseDB */
int searchClauseDB(head, bodies)
Pred* head;
Stack* bodies;
{
int i, j;

for(i = 0; i < clauseDB.DBSize; i++){
if(!termcmp(head, clauseDB.elem[i]->head)){
if(cmpAryStk(clauseDB.elem[i]->bodies, clauseDB.elem[i]->size, bodies)) return(0);
return(i + 1);
}
}
return(0);
}

/* int searchTmpClauseDB(Pred*, Stack*) search a clause which has same head and bodies
on TmpClauseDB */
int searchTmpClauseDB(head, bodies)
Pred* head;
Stack* bodies;
{
int i, j;

for(i = 0; i < tmpClauseDB.DBSize; i++){
if(!termcmp(head, tmpClauseDB.elem[i]->head)){
if(cmpAryStk(tmpClauseDB.elem[i]->bodies, tmpClauseDB.elem[i]->size, bodies)) return(0);
return(i + 1);
}
}
return(0);
}

/* int varID(Var*) looks for ID number of Var* */
int varID(var)
Var* var;
```

```
{
int i;
for(i = 0; i < varDB.DBSize; i++)
if(!namecmp(var->name, varDB.elem[i]->name))
return(i + 1);
return(0);
}

/* procedures for copying */

/* IStack* copyIStack(IStack*) is copy area of istack */
IStack* copyIStack(istack)
IStack* istack;
{
IStack* top;
IStack* rear;
IStack* index;
rear = allocIStack();
for(top = rear, index = istack; index->pre; top = top->pre, index = index->pre){
top->this = index->this;
top->truth = index->truth;
top->bodyTruth = index->bodyTruth;
if(index->hedge){
top->hedge = (Hedge*) malloc(sizeof(Hedge) * (strlen(index->hedge) + 1));
strcpy(top->hedge, index->hedge);
}
top->pre = allocIStack();
}
top->this = index->this;
top->truth = index->truth;
top->bodyTruth = index->bodyTruth;
if(index->hedge){
top->hedge = (Hedge*) malloc(sizeof(Hedge) * (strlen(index->hedge) + 1));
strcpy(top->hedge, index->hedge);
}
top->pre = NULL;
return(rear);
}

/* Term* copyTerm(Term*) makes deep copy */
Term* copyTerm(origin)
Term* origin;
{
Term* destination;
int i;
if(tmpTermDB.DBSize >= TMPTERMDBLIMIT){
fprintf(yyout,"Memory over flow. no more garantee\n");
fprintf(yyout,"Please do dbclean or restart lbfp system.\n");
}
destination = allocTerm();
destination->name = origin->name;
destination->size = origin->size;
destination->hedge = origin->hedge;
/*if(destination->name->flag == TMPVAR)
destination->name->flag = TMPVAR - 1;*/
```

```
if(origin->size) destination->terms = (Term**) malloc(sizeof(Term*) * origin->size);
for(i = 0; i < destination->size; i++)
destination->terms[i] = copyTerm(origin->terms[i]);
tmpTermDB.elem[tmpTermDB.DBSize++] = destination;
return(destination);
}

/* Clause* copyClause(Clause*) makes deep copy */
Clause* copyClause(origin)
Clause* origin;
{
Clause* destination;
Stack* vars;
int i;
void tmpVar();
Stack* seekVar();

vars = allocStack();
destination = allocClause();
destination->size = origin->size;
if(origin->head){
destination->head = copyTerm(origin->head);
seekVar(&(destination->head), vars);
}
destination->bodies = (Pred**) malloc(sizeof(Pred*) * origin->size);
for(i = 0; i < destination->size; i++){
destination->bodies[i] = copyTerm(origin->bodies[i]);
seekVar(destination->bodies + i, vars);
}
if(origin->head) tmpVar(destination->head, USETMP);
for(i = 0; i < destination->size; i++)
tmpVar(destination->bodies[i], USETMP);
destination->truth = origin->truth;
destination->certainty = origin->certainty;
return(destination);
}

/* procedures for converting */

/* Stack* stack?????(Term*) converts Term into Stack such as next=NULL */

Stack* stackTerm(item)
Term* item;
{
Stack* stack;
stack = allocStack();
stack->this = item;
return(stack);
}

#define stackPredicate stackTerm

/* Term* list(Stack*) converts Stack-formed list into Term-form */
Term* list(stack)
Stack* stack;
```

```
{
Term* term;

if(!stack) return(emptyList);
term = allocTerm();
term->name = listName;
term->size = 2;
term->terms = (Term**) malloc(sizeof(Term*) * 2);
term->terms[0] = stack->this;
term->terms[1] = list(stack->next);
return(term);
}

/* procedures for recording DB /

/* Term* record?????(Name*(, Stack*)) records argument on DB */
Term* recordTerm(name)
Name* name;
{
Term* term;

if(term = searchTermDB(name, NULL)){
free(name);
return(term);
}
if(termDB.DBSize < TERMDBLIMIT){
term = allocTerm();
term->name = name;
termDB.elem[termDB.DBSize++] = term;
return(term);
} else {
printf("Data base is full\n");
return(NULL);
}
}

Term* recordVariable(name)
Name* name;
{
Var* var;

if(var = searchVarDB(name)){
free(name);
return(var);
}
if(varDB.DBSize < TERMDBLIMIT){
var = allocVar();
var->name = name;
varDB.elem[varDB.DBSize++] = var;
return(var);
} else {
printf("Data base is full\n");
return(NULL);
}
}
```

```
Term* recordFunction(name, terms)
Name* name;
Stack* terms;
{
Term* term;
int argc;

if(term = searchTermDB(name, terms)){
free(name);
return(term);
}
if(termDB.DBSize < TERMDBLIMIT){
argc = depth(terms);
term = allocTerm();
term->name = name;
term->size = argc;
term->terms = allocTerms(argc, terms);
termDB.elem[termDB.DBSize++] = term;
return(term);
} else {
printf("Data base is full\n");
return(NULL);
}
}

Pred* recordPredicate(hedge, name, terms)
Hedge* hedge;
Name* name;
Stack* terms;
{
Pred* pred;
int argc;
void reduceHedge();

if(hedge) reduceHedge(hedge);
if(pred = searchPredDB(hedge, name, terms)){
/* free(hedge);
free(name);
free(terms);*/
return(pred);
}
if(predDB.DBSize < TERMDBLIMIT){
argc = depth(terms);
pred = allocPred();
if(hedge){
pred->hedge = hedge;
} else {
pred->hedge = (Hedge*) malloc(sizeof(Hedge) * 2);
pred->hedge[0] = '=';
pred->hedge[1] = NULL;
}
pred->name = name;
pred->size = argc;
pred->terms = allocTerms(argc, terms);
```

```
predDB.elem[predDB.DBSize++] = pred;
return(pred);
} else {
printf("Data base is full\n");
return(NULL);
}
}

/* int recordClause(Pred*, Stack*, double, double) records argument on DB */
int recordClause(head, bodies, certainty, truth)
Pred* head;
Stack* bodies;
double truth, certainty;
{
Clause* clause;
int i, id, size;
void tmpVar();

if(id = searchClauseDB(head, bodies)) return(id);
if(clauseDB.DBSize < CLAUSEDBLIMIT){
size = depth(bodies);
clause = allocClause();
clause->head = head;
clause->size = size;
clause->bodies = allocTerms(size, bodies);
clause->truth = truth;
clause->certainty = certainty;
tmpVar(clause->head, !USETMP);
for(i = 0; i < size; i++)
tmpVar(clause->bodies[i], !USETMP);

clauseDB.elem[clauseDB.DBSize++] = clause;
#ifdef FPRODEBUG
printClause(clause);
#endif
return(clauseDB.DBSize);
} else {
printf("Data base is full\n");
return(0);
}
}

/* int recordTmpClause(Pred*, Stack*, double, double) records argument on DB */
int recordTmpClause(head, bodies, certainty, truth)
Pred* head;
Stack* bodies;
double truth, certainty;
{
Clause* clause;
int i, id, size;
void tmpVar();
if(id = searchTmpClauseDB(head, bodies)) return(id);
if(tmpClauseDB.DBSize < TMPCLAUSEDBLIMIT){
size = depth(bodies);
clause = allocClause();
```

```
clause->head = head;
clause->size = size;
clause->bodies = allocTerms(size, bodies);
clause->truth = truth;
clause->certainty = certainty;
tmpVar(clause->head, !USETMP);
for(i = 0; i < size; i++)
tmpVar(clause->bodies[i], !USETMP);

tmpClauseDB.elem[tmpClauseDB.DBSize++] = clause;
#ifdef FPRODEBUG
printClause(clause);
#endif
return(tmpClauseDB.DBSize);
} else {
printf("Data base is full\n");
return(0);
}
}

/* procedures for memory free */

/* void freeIStack(IStack*) is free area of istack */
void freeIStack(istack)
IStack* istack;
{
IStack* index;
for(index = istack; index; istack = index->pre, index->pre = NULL, free(index), index
= istack);
}

/* void freeStack(Stack*) is free area of stack */
void freeStack(stack)
Stack* stack;
{
Stack* index;
for(index = stack; index; stack = index->next, index->next = NULL, index->this = NULL,
free(index), index = stack);
}

/* void freeSubstitute(Substitute*) is free area of substitute */
void freeSubstitute(sub)
Substitute* sub;
{
Substitute* index;
for(index = sub; index; sub = index->next, index->next = NULL, index->var = NULL, index->
= NULL, free(index), index = sub);
}

/* procedures for redundant */

/* void checkRedundant(Term**, int) checks term reduntancy and omits them */
void checkRedundant(terms, size)
Term** terms;
int size;
```

```
{
int i, index;
for(i = 0; i < size; i++)
for(index = i; index < size; index++)
if(!termcmp(terms[index],terms[i])){
/*free(terms[i]);*/
terms[i] = terms[index];
}
}

/* void omitRedundant(Substitute*) omits redundant substitutes */
void omitRedundant(subs)
Substitute* subs;
{
Substitute* index1;
Substitute* index2;
Substitute* tmp;
index1 = subs;
do{
if(!index1->next) break;
index2 = index1;
do{
if(!index1->var || !index2->next->var){
index2 = index2->next;
continue;
}
if(!termcmp(index1->var, index2->next->var) && !termcmp(index1->target, index2->next->tar
tmp = index2->next;
index2->next = index2->next->next;
tmp->var = NULL;
tmp->target = NULL;
tmp->next = NULL;
free(tmp);
} else{
index2 = index2->next;
}
} while(index2->next);
if(index1->next) index1 = index1->next;
} while(index1->next);
}

/* procedures for Truth value */

/* float inter01(Name*) makes truth value or certainty value in [0,1] */
double inter01(name)
Name* name;
{
double num = atof(name->string);
num = num > 1.0 ? 1.0 : num;
num = num < 0.0 ? 0.0 : num;
return(num);
}

/* double bp(double,double) is Bounded product */
double bp(a,b)
```

```
double a,b;
{
return(MAX( a + b -1.0 ,0 ));
}

/* double hedge(double,Hedge*) is hedge operation of h*t */
double hedge(t,h)
double t;
Hedge* h;
{
double ht;
int i;

ht = t;
if (!h) return(ht);
for( i = 0; i < strlen(h); i++){
switch(h[i]){
case(NOT): ht = 1 - ht;
break;
case(VERY): ht = pow(ht, 2.0);
break;
case(MORL): ht = sqrt(ht);
break;
case(PLUS): ht = pow(ht, 1.25);
break;
case(MINUS): ht = pow(ht, 0.8);
break;
case(EXACT): ht = (ht == 1.0) ? 1.0 : 0.0;
break;
default:  break;
}
}
return(ht);
}

/* procedures for Stack */

/* int depth(Stack*) accounts depth of stack */
int depth(stack)
Stack* stack;
{
int size = 0;
Stack* index;
for(index = stack; index; size++, index = index->next);
return(size);
}

/* Stack* add?????(Stack*, Stack*) links second stack to first stack */
Stack* addTerms(first, second)
Stack* first;
Stack* second;
{
Stack* index;
for(index = first; index->next ;index = index->next);
index->next = second;
```

```
return(first);
}

#define addPredicates addTerms

/* procedures for Temporary terms */

/* Name* tmpVarName(Name*) makes temporaly var name */
Name*  tmpVarName(name)
Name* name;
{
Name* tmpName;
char* string;
string = (char*) malloc(sizeof(char) * 8);
if(name->flag == TMPVAR) return(name);
tmpName = allocName();
sprintf(string,"_%d",tmpTermDB.TmpVarIndex++);
tmpName->flag = TMPVAR;
tmpName->string = string;
return(tmpName);
}

/* void tmpVar(Term*, flag) exchanges variables to temp name */
/* flg = USETMP then renames variables and !USETMP uses original names */
void tmpVar(term, flg)
Term* term;
int flg;
{
int i;
if(term->name->flag && flg){
term->name = tmpVarName(term->name);
} else {
for(i = 0; i < term->size; i++)
tmpVar(term->terms[i], flg);
}
}

/* void cleanTmpDB() cleans tmpClauseDB */
void cleanTmpDB()
{
int i;
char in;
fprintf(yyout,"Confirm: really clean up Temporary facts?[n]: ");
in = fgetc(stdin);
if(in == 'y'|| in == 'Y'){
for(i = 0; i < tmpClauseDB.DBSize; i++){
tmpClauseDB.elem[i]->head = NULL;
tmpClauseDB.elem[i]->bodies = NULL;
free(tmpClauseDB.elem[i]);

tmpClauseDB.DBSize = 0;
tmpClauseDB.DBIndex = 0;
fprintf(yyout,"finished cleaning up Temporary facts\n");
}
} else {
```

```
fprintf(yyout,"Rejected cleaning up Temporary facts\n");
}
for(;fgetc(stdin) != '\n';);
}

/* void refreshTmpTerm() cleans up tmpTermDB and tmpVarIndex */
void refreshTmpTerm()
{
int i;
for(i = 0; i < tmpTermDB.DBSize; i++){
tmpTermDB.elem[i]->name = NULL;
tmpTermDB.elem[i]->terms = NULL;
free(tmpTermDB.elem[i]);
}
tmpTermDB.DBSize = 0;
tmpTermDB.TmpVarIndex = 0;
}

/* procedures for List */

/* Term* listCarCdr(Term*, Term*) makes list from car and cdr */
Term* listCarCdr(car, cdr)
Term* car;
Term* cdr;
{
Term* term;

term = allocTerm();
term->name = listName;
term->size = 2;
term->terms = (Term**) malloc(sizeof(Term*) * 2);
term->terms[0] = car;
term->terms[1] = cdr;
return(term);
}

/* procedures for Hedge */

/* Pred* attachHedge(Hedge*, Pred*) attaches a hedge to a predicate */
/*Pred* attachHedge(hedge, pred)
Hedge* hedge;
Pred* pred;
{
Hedge* h;
h = pred->hedge;
pred->hedge = hedge;
free(h);
return(pred);
}*/

/* Hedge* addHedge(Hedge, Hedge*) convert hedge to string */
Hedge* addHedge(h, hedge)
Hedge h;
Hedge* hedge;
{
```

```
Hedge* newHedge;
int len;

if (hedge){
len = strlen(hedge);
if (h == NORMAL){
return(hedge);
}
if (len == 1 && hedge[len - 1] == NORMAL){
hedge[len - 1] = h;
return(hedge);
}
} else {
len = 0;
}
newHedge = (Hedge*) malloc((sizeof(Hedge)) * (len + 2 ));
if (len){
strcpy(newHedge, hedge);
free(hedge);
}
newHedge[len] = h;
newHedge[len + 1] = NULL;
return(newHedge);
}

/* Hedge encodeHedge(Name*) convert hedge to string */
Hedge encodeHedge(name)
Name* name;
{
Hedge h;

switch(name->string[0]){
case('n'):  h = NOT;
break;
case('v'):  h = VERY;
break;
case('m'):  h = name->string[1] == 'o' ? MORL : MINUS;
break;
case('p'):  h = PLUS;
break;
case('e'):  h = EXACT;
break;
default:  h = NORMAL;
break;
}
return(h);
}

/* void reduceHedge(Hedge*) reduces redundant hedge */
void reduceHedge(hedge)
Hedge* hedge;
{
Hedge* newHedge;
int i, j, len;
Hedge* exactIndex;
```

```
len = strlen(hedge);
newHedge = (Hedge*) malloc(sizeof(Hedge) * (len + 1));
j = 0;
exactIndex = strchr(hedge, EXACT);
for ( i = 0; i < len; i++ ){
if(exactIndex){
switch(hedge[i]){
case(NOT): if( j == 0 ){
newHedge[j++] = NOT;
} else if(newHedge[j - 1] == NOT){
j--;
} else {
newHedge[j++] = NOT;
}
break;
case(EXACT): if(hedge + i == exactIndex)
newHedge[j++] = EXACT;
break;
default:
break;
}
continue;
} else {
if ( j == 0 ) {
newHedge[j++] = hedge[i];
continue;
}
switch(hedge[i]){
case(NOT): if(newHedge[j - 1] == NOT){
j--;
} else {
newHedge[j++] = NOT;
}
break;
case(VERY): if(newHedge[j - 1] == MORL){
j--;
} else {
newHedge[j++] = VERY;
}
break;
case(MORL): if(newHedge[j - 1] == VERY){
j--;
} else {
newHedge[j++] = MORL;
}
break;
case(PLUS): if(newHedge[j - 1] == MINUS){
j--;
} else {
newHedge[j++] = PLUS;
}
break;
case(MINUS): if(newHedge[j - 1] == PLUS){
j--;
```

```
} else {
newHedge[j++] = MINUS;
}
break;
default:
break;
}
}
}
newHedge[j] = NULL;
strcpy(hedge,newHedge);
if (j < len )
free(hedge + j + 1);
hedge[j] = NULL;
free(newHedge);
}

/* Hedge* antHedge(Hedge*) makes antonomy of hedge */
Hedge* antHedge(hedge)
Hedge* hedge;
{
int i,len;
Hedge* ant;

len = strlen(hedge);
ant = (Hedge*) malloc(sizeof(Hedge) * (len + 1));
for ( i = 0; i < len; i++){
switch(hedge[len - i - 1]){
case(NOT) : ant[i] = NOT;
break;
case(VERY) : ant[i] = MORL;
break;
case(MORL) : ant[i] = VERY;
break;
case(PLUS) :  ant[i] = MINUS;
break;
case(MINUS) : ant[i] = PLUS;
break;
case(EXACT) : ant[i] = ANTEXACT;
break;
default :  break;
}
}
return(ant);
}

/* Hedge* divHedge(Hedge*, Hedge*) makes division hedge and hedge */
Hedge* divHedge(uHedge, bHedge)
Hedge* uHedge;
Hedge* bHedge;
{
Hedge* newHedge;

newHedge = (Hedge*) malloc(sizeof(Hedge) * (strlen(uHedge) + strlen(bHedge)) + 1);
strcpy(newHedge, antHedge(bHedge));
```

```
strcat(newHedge, uHedge);
reduceHedge(newHedge);
return(newHedge);
}

/* procedures for Built-in predicates */

/* void listAllClauses() print all clauses */
void listAllClauses()
{
int i;
for(i = 0; i < tmpClauseDB.DBSize; i++)
printClause(tmpClauseDB.elem[i]);
for(i = 0; i < clauseDB.DBSize; i++)
printClause(clauseDB.elem[i]);
}

/* void listMatchedClauses(Name) prints clauses matched with Name. */
void listMatchedClauses(name)
Name* name;
{
int i;
for(i = 0; i < tmpClauseDB.DBSize; i++)
if (!namecmp(name ,tmpClauseDB.elem[i]->head->name))
printClause(tmpClauseDB.elem[i]);
for(i = 0; i < clauseDB.DBSize; i++)
if (!namecmp(name ,clauseDB.elem[i]->head->name))
printClause(clauseDB.elem[i]);
}

/* void setIn(char*) sets yyin */
void setIn(fname)
char* fname;
{
if(fname)
if(!(yyin = fopen(fname,"r"))){
fprintf(yyout,"File %s not found.\n", fname);
yyin = stdin;
}
}

/* void cleanDB() cleans up all DB */
void cleanDB()
{
int i;
char in;
fprintf(yyout,"Confirm: really clean up DB?[n]: ");
in = fgetc(stdin);
if(in == 'y'|| in == 'Y'){
for(i = 0; i < tmpTermDB.DBSize; i++){
tmpTermDB.elem[i]->name = NULL;
tmpTermDB.elem[i]->terms = NULL;
free(tmpTermDB.elem[i]);
}
for(i = 0; i < tmpClauseDB.DBSize; i++){
```

```
tmpClauseDB.elem[i]->head = NULL;
tmpClauseDB.elem[i]->bodies = NULL;
free(tmpClauseDB.elem[i]);
}
for(i = 0; i < termDB.DBSize; i++){
termDB.elem[i]->name = NULL;
termDB.elem[i]->terms = NULL;
free(termDB.elem[i]);
}
for(i = 0; i < varDB.DBSize; i++){
varDB.elem[i]->name = NULL;
varDB.elem[i]->terms = NULL;
free(varDB.elem[i]);
}
for(i = 0; i < predDB.DBSize; i++){
predDB.elem[i]->name = NULL;
predDB.elem[i]->terms = NULL;
free(predDB.elem[i]);
}
for(i = 0; i < clauseDB.DBSize; i++){
clauseDB.elem[i]->head = NULL;
clauseDB.elem[i]->bodies = NULL;
free(clauseDB.elem[i]);
}
initDB();
fprintf(yyout,"finished cleaning up DB\n");
} else {
fprintf(yyout,"Rejected cleaning up DB\n");
}
for(;fgetc(stdin) != '\n';);
}

/* CStack* checkBuiltInPred(Pred*) checks and operates of builtIn predicate */
CStack* checkBuiltInPred(pred)
Pred* pred;
{
int i;
void cleanDB();
void cleanTmpDB();
if(!pred) return(emptyStack);
if(!strcmp(pred->name->string, "list")){
if (pred->size){
for(i = 0; i < pred->size; i++)
listMatchedClauses(pred->terms[i]->name);
} else{
listAllClauses();
}
return(emptyStack);
}
if(!strcmp(pred->name->string, "consult")){
setIn(pred->terms[0]->name->string);
return(emptyStack);
}
if(!strcmp(pred->name->string, "dbstat")){
fprintf(yyout,"termDB capacity = %f %%\n",(float)termDB.DBSize/(float)TERMDBLIMIT*100.0);
```

```
fprintf(yyout,"varDB capacity = %f %%\n",(float)varDB.DBSize/(float)TERMDBLIMIT*100.0);
fprintf(yyout,"predDB capacity = %f %%\n",(float)predDB.DBSize/(float)TERMDBLIMIT*100.0);
fprintf(yyout,"clauseDB capacity = %f %%\n",(float)clauseDB.DBSize/(float)CLAUSEDBLIMIT*1
fprintf(yyout,"tmpTermDB capacity = %f %%\n",(float)tmpTermDB.DBSize/(float)TMPTERMDBLIMI
fprintf(yyout,"tmpClauseDB capacity = %f %%\n",(float)tmpClauseDB.DBSize/(float)CLAUSEDBL
return(emptyStack);
}
if(!strcmp(pred->name->string, "dbclean")){
cleanDB();
return(emptyStack);
}
if(!strcmp(pred->name->string, "tmpclean")){
cleanTmpDB();
return(emptyStack);
}
if(!strcmp(pred->name->string, "quit")){
fprintf(yyout,"FProlog/LbFP %s is terminated.\n",version);
fclose(yyout);
exit(1);
}
return(NULL);
}

/* procedures for Substitute */

/* Substitute* addSubstitute(SubStitute**, Substitute*) checks confliction among substitu
*/
Substitute* addSubstitute(top, sub)
Substitute** top;
Substitute* sub;
{
Substitute* index;

if(!*top)return (*top = sub);
if(!(*top)->var)return (*top = sub);
if(!sub)return (*top);
if(!sub->var)return (*top);
for(index = *top; index->next; index = index->next)
if(!termcmp(index->var, sub->var) && termcmp(index->target, sub->target))
return(NULL);
if(!termcmp(index->var, sub->var) && termcmp(index->target, sub->target))
return(NULL);
index->next = sub;
return(*top);
}

/* void replace(Term**, Substitute*) replaces all variables */
void replace(term, sub)
Term** term;
Substitute* sub;
{
int i;
Substitute* index;

if(!sub->var){
```

```
} else if((*term)->name->flag){
for(index = sub; index->next  && namecmp(index->var->name, (*term)->name); index =index->
if(!namecmp(index->var->name, (*term)->name)){
*term = allocTerm();
(*term)->name = index->target->name;
(*term)->size = index->target->size;
(*term)->terms = index->target->terms;
/* these are now testing */
if ((*term)->name->flag)
replace(term, sub);
for(i = 0; i < (*term)->size; i++)
replace((*term)->terms + i, sub);
/* */
}
} else {
for(i = 0; i < (*term)->size; i++)
replace((*term)->terms + i, sub);
}
/* these are now testing */
/**/
}

/* void doSubstitute(CStack*) substitutes variables */
void doSubstitute(target)
CStack* target;
{
int i;
Clause* resolvent;
resolvent = target->this;
/*
free(resolvent->head);
resolvent->head = NULL;*/
for(i = 0; i < resolvent->size; i++)
replace(resolvent->bodies + i,target->substitute);
target->substitute = NULL;
/*target->this = resolvent;*/
}

/* Substitute* combineSubstitute(Substitute*, Substitute*) */
Substitute* combineSubstitute(preSubs,subs)
Substitute* preSubs;
Substitute* subs;
{
Substitute* preIndex;
Substitute* index;
Substitute* newSubs;
Substitute* top;
if(!subs) return(preSubs);
if(!subs->var) return(preSubs);
top = allocSubstitute();
newSubs = top;
for(preIndex = preSubs; preIndex; preIndex = preIndex->next){
if(preIndex->var){
newSubs->var = copyTerm(preIndex->var);
newSubs->target = copyTerm(preIndex->target);
```

```
newSubs->next = allocSubstitute();
replace(&(newSubs->target), subs);
newSubs = newSubs->next;
}
}
if(!top->next){
top = subs;
return(top);
}
for(newSubs = top; newSubs->next->next; newSubs = newSubs->next);
freeSubstitute(newSubs->next);
newSubs->next = subs;
return(top);
}

/* procedures for matching */

/* Substitute* termMatch(Term*, Term*) tests whether both terms can be match or not and
makes largest substitute */
Substitute* termMatch(term1, term2)
Term* term1;
Term* term2;
{
int i;
Substitute* sub;
Substitute* top;
top = allocSubstitute();
if(term1->name->flag){
top->var = term1;
top->target = term2;
return(top);
} else if(term2->name->flag){
top->var = term2;
top->target = term1;
return(top);
}
if(term1->size != term2->size) return(NULL);
if(namecmp(term1->name, term2->name)) return(NULL);
for(i = 0; i < term1->size; i++){
if(!(sub = termMatch(term1->terms[i], term2->terms[i])))
return(NULL);
if(!addSubstitute(&top,sub)){
/*freeSubstitute(top);*/
/*freeSubstitute(sub);*/
return(NULL);
}
}
return(top);
}

/* CStack* match(Clause*, int*, IStack*) looks for a matchable clause from DB */
CStack* match(goal, start, istack)
Clause* goal;
int* start;
IStack* istack;
```

```
{
int i, index;
double tmpTruth;
Substitute* subs;
CStack* target;
Clause* newClause;
Clause* indexC;
IStack* nextI;
Clause* copyClause();
Term* copyTerm();
if(!goal->size) return(NULL);
/* search candidacy on built in Clause DB */
if(target = checkBuiltInPred(goal->bodies[0])){
if(goal->size == 1) return(target);
target = allocCStack();
newClause = target->this = allocClause();
newClause->size = goal->size - 1;
newClause->bodies = (Pred**) malloc(sizeof(Pred*) * newClause->size);
for(i = 1; i < goal->size; i++){
newClause->bodies[i - 1] = copyTerm(goal->bodies[i]);
}
newClause->truth = goal->truth;
newClause->certainty = goal->certainty;
target->substitute = allocSubstitute();
target->istack = copyIStack(istack);
target->istack->this--;
target->truth = newClause->truth;
target->certainty = newClause->certainty;
return(target);
}
if (*start >= clauseDB.DBSize + tmpClauseDB.DBSize)
return(NULL);
/* search candidacy on tmpClauseDB */
for(i = *start ; i < tmpClauseDB.DBSize; i++){
if(roughTermcmp(goal->bodies[0], tmpClauseDB.elem[i]->head))
continue; /* NG on roughTermcmp()*/
if(goal->certainty > tmpClauseDB.elem[i]->certainty)
continue; /* NG by alpha(certainty)-cut on DB */
tmpTruth = bp(istack->truth, tmpClauseDB.elem[i]->truth);
for(nextI = istack; tmpTruth > 0.0 && nextI->pre; nextI = nextI->pre)
tmpTruth = bp(tmpTruth, nextI->pre->truth);
if(tmpTruth <= 0.0)
continue; /* NG by meaningless derivation */
indexC = copyClause(tmpClauseDB.elem[i]);
#ifdef FPRODEBUG
printClause(indexC);
#endif
if(subs = termMatch(goal->bodies[0], indexC->head))
break;
}
/* search candidacy on clauseDB */
for(; i >= tmpClauseDB.DBSize && i < clauseDB.DBSize + tmpClauseDB.DBSize; i++){
if(roughTermcmp(goal->bodies[0], clauseDB.elem[i - tmpClauseDB.DBSize]->head))
continue; /* NG on roughTermcmp()*/
if(goal->certainty > clauseDB.elem[i - tmpClauseDB.DBSize]->certainty)
```

```
continue; /* NG by alpha(certainty)-cut on DB */
tmpTruth = bp(istack->truth, clauseDB.elem[i - tmpClauseDB.DBSize]->truth);
for(nextI = istack; tmpTruth > 0.0 && nextI->pre; nextI = nextI->pre)
tmpTruth = bp(tmpTruth, nextI->pre->truth);
if(tmpTruth <= 0.0)
continue; /* NG by meaningless derivation */
indexC = copyClause(clauseDB.elem[i - tmpClauseDB.DBSize]);
#ifdef FPRODEBUG
printClause(indexC);
#endif
if(subs = termMatch(goal->bodies[0], indexC->head))
break; /* GOOD on strict matching (if NG then continue)*/
}

index = i;
if(index >= tmpClauseDB.DBSize + clauseDB.DBSize) return(NULL);
target = allocCStack(); /* make resolvent CStack */
newClause = target->this = allocClause(); /* make resolvent (derivated clause) */
newClause->size = indexC->size + goal->size - 1;
newClause->bodies = (Pred**) malloc(sizeof(Pred*) * newClause->size);
for(i = 0; i < indexC->size; i++)
newClause->bodies[i] = indexC->bodies[i];
for(i = 1; i < goal->size; i++){
newClause->bodies[indexC->size + i - 1] = copyTerm(goal->bodies[i]);
/*tmpVar(newClause->bodies[indexC->size + i - 1],USETMP);*/
}
newClause->truth = goal->truth;
if(goal->certainty > 0.0){
newClause->certainty = goal->certainty;
} else {
newClause->certainty = MAX(goal->certainty, VARIABLE * indexC->certainty);
}
target->substitute = subs;
target->truth = newClause->truth;
target->certainty = newClause->certainty;
target->istack = copyIStack(istack);
target->istack->this--;
if(indexC->size){ /* make and stack new IStack (remaining sub derivation tree)*/
nextI = allocIStack();
nextI->this = indexC->size;
nextI->truth = indexC->truth;
nextI->hedge = divHedge(goal->bodies[0]->hedge, indexC->head->hedge);
nextI->bodyTruth = 1.0;
nextI->pre = target->istack;
target->istack = nextI;
} else{ /* calculate IStack's tmpTruth (made sub derivation tree completely)*/
target->istack->bodyTruth = MIN(target->istack->bodyTruth, hedge(indexC->truth,  divHedge
indexC->head->hedge)));
}
*start = index + 1;
return(target);
}

/* Stack* seekVar(Term**, Stack*) checks same name variable */
Stack* seekVar(term, stack)
```

```
Term** term;
Stack* stack;
{
Stack* index;
int i;
if((*term)->name->flag){
for(index = stack; index->this; index = index->next)
if(!termcmp(index->this, *term)){
*term = index->this;
return(stack);
}
index->this = *term;
index->next = allocStack();
return(stack);
} else {
for(i = 0; i < (*term)->size; i++)
seekVar((*term)->terms + i, stack);
return(stack);
}
}

/* procedures for resolution */

/* void refutate(CStack*) accepts refutation */
void refutate(target)
CStack* target;
{
CStack* index;
Substitute* subs;
Clause* subResolve();
subs = target->substitute;
for(index = target; index->pre; index = index->pre){
subs = combineSubstitute(index->pre->substitute,subs);
#ifdef FPRODEBUG
printSubstitute(subs);
#endif
}
subs = combineSubstitute(subs,subs);
if(target->truth < 1.0 || target->certainty < 1.0)
subResolve(target,subs);
if(index->this->certainty <= VARIABLE){
printTerm(varDB.elem[(int) (VARIABLE * index->this->certainty) - 1]);
fprintf(yyout," = %8.6f\n",target->certainty);
}
omitRedundant(subs);
printSubstitute(subs);
fprintf(yyout,"[%8.6f]\n",target->truth);
}

/* int askMore() asks need one more answer or not */
/* if answer is YES then 0 NO then 1 */
int askMore()
{
char in;
char moreview();
```

```
/*
if((in = fgetc(stdin)) == '\n') return(1);
for(;fgetc(stdin) != '\n';);
if(in == ';') return(0);
*/
fprintf(yyout,"\n");
if (moreview() == ';') return (0);
return(1);
}

/* Clause* subResolve(CStack*, Substitute*) dose sub resolution */
Clause* subResolve(subTop, subs)
CStack* subTop;
Substitute* subs;
{
Clause* subGoal;
CStack* top;
CStack* sindex;
CStack* resolvent;
IStack* istack;
IStack* nextIStack;
Stack* vars;
int status = YES;
int i, level, size;
for(top = subTop; top->pre; top = top->pre);
subGoal = allocClause();
subGoal->size = top->this->size;
subGoal->bodies = (Pred**) malloc(sizeof(Pred*) * subGoal->size);
for(i = 0; i < subGoal->size; i++){
subGoal->bodies[i] = copyTerm(top->this->bodies[i]);
replace(subGoal->bodies + i, subs);
}
subGoal->truth = 1.0;
subGoal->certainty = subTop->certainty > 0.0 ? subTop->certainty : VARIABLE;

top = allocCStack();
top->this = subGoal;
top->pre = NULL;
top->index = 0;
top->truth = subTop->truth;
top->certainty = subTop->certainty = subGoal->certainty;
top->istack = allocIStack();
top->istack->this = subGoal->size;
top->istack->bodyTruth = 1.0;
top->istack->truth = 1.0;
top->istack->pre = NULL;
sindex = top;
level = 1;
while(level){
#ifdef FPRODEBUG
fprintf(yyout,"[-%d]\n",level);
#endif
do{
if(!(resolvent = match(sindex->this, &sindex->index, sindex->istack))) break;
} while(subTop->truth > resolvent->istack->truth);
```

```
if(resolvent){
sindex->substitute = resolvent->substitute;
doSubstitute(resolvent);
if(resolvent == emptyStack) break;
for(; !resolvent->istack->this;){
if(resolvent->istack->pre){
resolvent->istack->pre->bodyTruth = MIN(resolvent->istack->pre->bodyTruth, hedge(bp(resol
resolvent->istack->truth), resolvent->istack->hedge));
istack = resolvent->istack->pre;
resolvent->istack->pre = NULL;
free(resolvent->istack);
resolvent->istack = istack;
} else{
subTop->truth = MAX(subTop->truth,  hedge(bp(resolvent->istack->bodyTruth, resolvent->ist
resolvent->istack->hedge));
break;
}
}
resolvent->pre = sindex;
sindex = resolvent;
if(emptyClause(resolvent)){
subTop->certainty = MAX(subTop->certainty, VARIABLE * resolvent->certainty);
}
level++;
} else {
resolvent = sindex->pre;
if(!resolvent) break;
freeIStack(sindex->istack);
sindex = resolvent;
level--;
}
}
}

/* Clause* resolve(Stack*, double, double) dose resolution */
Clause* resolve(goalS, certainty, truth)
Stack* goalS;
double truth, certainty;
{
Clause* goal;
CStack* top;
CStack* sindex;
CStack* resolvent;
IStack* istack;
Substitute* index;
int status = YES;
int i, level, size;
/* exchanges a stack to a goal clause */
size = depth(goalS);
goal = allocClause();
goal->head = NULL;
goal->size = size;
goal->bodies = allocTerms(size, goalS);
goal->truth = 0.0;
goal->certainty = certainty;
```

```
for(i =0; i < size; i++)
tmpVar(goal->bodies[i], !USETMP);

top = allocCStack();
top->this = goal;
top->pre = NULL;
top->index = 0;
top->truth = 0.0;
top->certainty = certainty;
top->istack = allocIStack();
top->istack->this = goal->size;
top->istack->truth = 1.0;
top->istack->bodyTruth = 1.0;
top->istack->pre = NULL;
sindex = top;
level = 1;
while(level){
#ifdef FPRODEBUG
fprintf(yyout,"[%d]\n",level);
#endif
if(resolvent = match(sindex->this, &sindex->index, sindex->istack)){
sindex->substitute = resolvent->substitute;
doSubstitute(resolvent);
if(resolvent == emptyStack) break;
#ifdef FPRODEBUG
printClause(resolvent->this);
for(index = sindex->substitute; index; index = index->next)
if(index->var){
printTerm(index->var);
fprintf(yyout," = ");
printTerm(index->target);
fprintf(yyout,":");
}
fprintf(yyout,"\n");

#endif
for(; !resolvent->istack->this;){
if(resolvent->istack->pre){
/* calculate truth value */
resolvent->istack->pre->bodyTruth = MIN(resolvent->istack->pre->bodyTruth, hedge(bp(resol
resolvent->istack->truth),resolvent->istack->hedge));
istack = resolvent->istack->pre;
resolvent->istack->pre = NULL;
free(resolvent->istack);
resolvent->istack = istack;
} else{
resolvent->truth = hedge(bp(resolvent->istack->bodyTruth, resolvent->istack->truth), reso
break;
}
}
resolvent->pre = sindex;
sindex = resolvent;
if(emptyClause(resolvent)){
refutate(resolvent);
if(askMore()){
```

```
fprintf(yyout,"Yes.\n");
break;
}
}
level++;
} else {
resolvent = sindex->pre;
if(resolvent){
/*free(sindex->this);
freeSubstitute(sindex->substitute);*/
freeIStack(sindex->istack);
sindex = resolvent;
} else {
status = NO;
fprintf(yyout,"No.\n");
break;
}
level--;
}
}
refreshTmpTerm();
}

/* procedures for main */

/* flag for wait for child setting */
extern int wait_flag;

void signal_child(signum)
int signum;
{
int *status;
if(signum == SIGUSR1){
wait_flag = 0;
} else {
fprintf(stderr,"Can't open view.\n",getpid());
yyin = stdin;
yyout = stdout;
}
}

int main(argc,argv)
int argc;
char* argv[];
{
int pipefd[2];
int* status;
pid_t pid;
void viewChild();

initDB();
initConst();
wait_flag = 1;
if( pipe(pipefd) < 0 ){
perror("pipe");
```

```
exit(1);
}
pid = fork();
if( pid > 0){
signal(SIGCHLD, signal_child);
signal(SIGUSR1, signal_child);
if( (yyout = fdopen(pipefd[1], "w")) < 0){
perror("fdopen");
kill(pid, SIGTERM);
exit(1);
}
if( (yyin = fdopen(pipefd[0], "r")) < 0){
perror("fdopen");
kill(pid, SIGTERM);
exit(1);
}
/* if(dup2(pipefd[1],fileno(yyout)) < 0){
perror("dup2");
kill(pid, SIGTERM);
exit(1);
}*/
}
if( pid == 0 ){
viewChild(&argc, argv, pipefd[0], pipefd[1]);
} else if(pid < 0){
yyout = stdout;
}
while(wait_flag);
if(argc == 2){
if(*argv[1] == '-'){
fprintf(stderr,"Usage: %s filename\n",argv[0]);
return(1);
} else if(yyin = fopen(argv[1],"r")){
} else{
fprintf(stderr, "preconsulting file open error\n");
yyin = stdin;
}
} else if (argc >= 3){
fprintf(stderr,"Usage: %s filename\n",argv[0]);
return(0);
}

fprintf(yyout,"Welcome to FProlog/LbFP %s\n",version);
yyparse();
}
```

D.2 字句解析部

D.2.1 fpro.lex

```
%{
```

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
void nameset();
%}

SPACE [ \t\n\r\f]
%%
{SPACE}+ ;

"/*".*"*/" ;

[a-z][a-zA-Z_0-9]* {
nameset(0);
return(NAME);
}

[A-Z_][a-zA-Z_0-9]* {
nameset(1);
return(VAR);
}

[0-9]+\.?[0-9]* {
nameset(0);
return(NUM);
}

\.[0-9]+ {
nameset(0);
return(NUM);
}

.  {
return(*yytext);
}

%%
/* set term's flag */
void nameset(f)
int f;
{
yylval.name = (Name*) malloc(sizeof(Name));
if(!(yylval.name->string = (char*) malloc(yyleng + 1)))
fprintf(stderr,"Name allocation is failed");
strcpy(yylval.name->string, yytext);
yylval.name->flag = f;
}
```

D.3 ヘッダファイル

D.3.1 fpro.h

```
#include <stdio.h>
```

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <math.h>

/* header file of LbFP */
#define TERMDBLIMIT 4096
#define TMPTERMDBLIMIT 65536
#define CLAUSEDBLIMIT 4096
#define TMPCLAUSEDBLIMIT 4096

#define VARIABLE -1.0

#define YES 1
#define NO 0

/* value of name->flag */
#define NONVAR 0
#define TMPVAR 2

/* use tmp variable flag */
#define USETMP 1

#define MAX(a,b) a>b?a:b
#define MIN(a,b) a<b?a:b

/* for hedge */
#define NORMAL '='
#define NOT '!'
#define VERY '^'
#define MORL '_'
#define PLUS '+'
#define MINUS '-'
#define EXACT '>'
#define ANTEXACT '<'

typedef char Hedge;

/* flag is set in nameset() in fpro.lex*/
/* 0 of NONVAR means a non variable term */
/* 1 means a variable term */
/* 2 of TMPVAR means a temporary variable term */
typedef struct {
char* string;
char flag;
} Name;

typedef struct STRUCTURE{
Name* name;
Hedge* hedge;
short int size;
struct STRUCTURE** terms;
} Term;

typedef struct STRUCTURE Pred;
typedef struct STRUCTURE Var;
```

```
typedef struct {
struct STRUCTURE* elem[TERMDBLIMIT];
short int DBSize;
} TermDB;

typedef TermDB  PredDB;
typedef TermDB  VarDB;

typedef struct {
Var* elem[TMPTERMDBLIMIT];
int DBSize;
int TmpVarIndex;
} TmpDB;

typedef struct {
Pred* head;
short int size;
Pred** bodies;
double truth;
double certainty;
} Clause;

typedef struct {
Clause* elem[CLAUSEDBLIMIT];
short int DBSize;
short int DBIndex;
} ClauseDB;

typedef struct {
Clause* elem[TMPCLAUSEDBLIMIT];
short int DBSize;
short int DBIndex;
} TmpClauseDB;

typedef struct SUBS{
Var* var;
Term* target;
struct SUBS* next;
} Substitute;

typedef struct STACK{
struct STRUCTURE* this;
struct STACK* next;
} Stack;

typedef struct ISTACK{
int this;
Hedge* hedge;
double truth;
double bodyTruth;
struct ISTACK* pre;
} IStack;

typedef struct CSTACK{
```

```
Clause* this;
Substitute* substitute;
int index;
IStack* istack;
double truth;
double certainty;
struct CSTACK* pre;
} CStack;

static TermDB termDB;
static TmpDB tmpTermDB;
static PredDB predDB;
static VarDB varDB;
static ClauseDB clauseDB;
static TmpClauseDB tmpClauseDB;
static Name* listName;
static Term* emptyList;
static CStack* emptyStack;

/* child pid */
static pid_t child_pid;
/* wait flag for child view setup (for parent process) */
static int wait_flag;
/* signal flag for read pipe (for chlid process) */
static int read_flag;
/* default label widget for dialog view */
static Widget defaultLabel;
/* for keeping LbFP's default input stream */
static FILE *lbfp_in;
```

D.3.2 version.h

```
static char version[] = "ver. 2.0.0";
```