修 士 論 文

否定的意味が内在するソートを含む非単調推論の研究

指導教官　東条敏 教授

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

水野幸喜

2001 年 2 月 15 日

要 旨

本稿では, 非単調推論の枠組みを使用し, 与えられた知識の間で成立する規則を生成するシステムを構築する. そして, 生成された規則間における階層関係を表現する. 次に強い否定をシステムに組み込むための手法について考察する.

従来の機能論理プログラミングは, 主にホーンプログラムや節形式プログラムにおける関係記述を行うものが多かった. しかし, この手法による記述法は単調な規則生成にしか扱えないものが多く, 例外処理やデフォルト規則を扱えるものは少ない. そのため, 例外処理やデフォルト規則を扱うことを可能とするために非単調推論の枠組みが必要となる.

規則生成において, 今回のような与える知識により生成される規則が変化するような状況では, 正例と負例の扱いに十分な考察が必要となる. その理由は, 正例とは一般的な常識として捉えた知識であり, 負例とは例外的なことがらという解釈をしているため, 今まで負例として扱っていた例が次の状況では, 正例として扱っていた例よりもより一般的な例となることが起こり得る. 規則生成システムに, ある事例に対してのすべての知識を与えることが可能であるならばこのような問題は発生しない. そして知識に不備があることから正例と負例の反転は引き起こされる. そして, その不備を補うために従来のシステムは閉世界仮説を適用する. 閉世界仮説とは記述されていないことは偽として扱うような理論である. 'Jack は鳥である' という知識があった場合を考える. Jack が飛ぶか飛ばないかの事実が記述されていない限り, 自動的に負例が採用されてしまう. 鳥は飛ぶことが一般的とされるようなシステムが構築されていたならば, この Jack は飛ばないとされてしまう. そして閉世界仮説により偽と決定されたこの例は, 後の規則生成時に負例の決定を受けているために, 鳥は飛ばないという規則を恒久的なものとしてしまう. このことは後に述べる例外処理において矛盾を発生させる要因となる. この矛盾を回避するためには, 閉世界仮説のような記述されていないことは偽とする理論の適用をやめ, 記述されていないことは無矛盾として扱える理論が必要となる. そのための理論として開世界特殊化が挙げられる. そしてこの理論を用いて正負例の反転を可能にするための考察を行う.

次に生成規則間に存在する階層関係を構築する. この階層構造は例外処理と深い関係を持つ. 例を挙げる. 鳥は一般的に飛ぶ, しかしダチョウやケガをした鳥は飛ばない. このように鳥を飛ぶという性質から観察すると, その集合の中に飛ばないような例外が存在する. 今, システムにおいて '鳥ならば飛ぶ' の規則が生成されているならば, '鳥ならば飛ばない' の規則は前者の規則の例外的な存在として階層構造をつくる. 次に動物について鳥

がどのような存在か考える. 動物は一般的に飛ばない. 例外として鳥が飛ぶ. 先ほど構成された階層構造が動物という枠の中に入り込んでいる. このような例外の例外処理を考察し階層の多重化を図る.

システムに否定的意味が内在する述語を含む知識を与え, この知識より規則生成を可能とする拡張を施す. 否定的意味が内在する表現として強い否定を採用する. 強い否定とは 'un' のような否定表現を持つものを指す. 強い否定は古典的否定よりも否定表現が強いとされる. '¬happy(a)' は a が幸せ以外であることを表すのに対し, 'unhappy(a)' は a が幸せでないと言い切ってしまう. 前者なら幸せであることのみを否定している. しかし後者は不幸であると解釈される. 否定表現の違いは指定する対象領域の変化をもたらす. この否定表現の違いによる対象領域の不一致の考察を行う.

規則生成システムに与える知識は一階の述語の形式であり例集合と背景知識からなる. 背景知識には事実を述語の形式で記述する. 例集合には正の述語と負の述語があり, 正の述語とは例集合に含まれる述語の肯定の形であり, 負の述語はその述語の否定の形である. 生成される規則は '例 $\Leftarrow$ 背景知識と 例外以外' の形をしている. 背景知識は実体の説明と解釈できる. 例えば 'bird(a)' は 'a は鳥である' を表す. 例は正の述語と負の述語がありそれぞれ 'fly+(b)', 'fly-(c)' は 'b は飛ぶ', 'c は飛ばない' を表す. システムの例集合中に 'X は飛ぶ' という知識が 'X は飛ばない' という知識を上回り, 背景知識に 'X は鳥' という知識がある状況で規則生成を行うと

'fly+(X) $\Leftarrow$ bird(X), \+abnormal(X)'

となる. 強い否定 'un' による述語は '¬' による述語の集合に包含されるようにシステムに組み込む. 例として

'unhappy(X) $\in$ ¬happy(X) の集合'

となるように規則生成を行う. この式を成立させるための解釈と条件について考察する. そしてこの手法により構成される階層がどのような構造となるか考察する.

目 次

第 1 章

はじめに

ILP（機能論理プログラミング）とは論理プログラミングの枠組みのなかで関係記述を行うものである. 従来の ILP では, 主に確定ホーンプログラムや節形式プログラムにおける関係記述を行なうものが多かった. しかし, ILP の研究には単調な規則生成にしか扱えないものが多く, 例外処理やデフォルト規則を扱えるものは少ない. AI での知識表現においては, 単調なクラスの論理に基づくプログラムは, 常識則や分類学的階層のような知識を表現するには不充分である. 従って, デフォルト規則を生成するメカニズムを実現するためには, 非単調推論を行える帰納推論の枠組みが必要になる.

今回の研究では失敗による否定（negation as failure；NAF）を含む知識表現を行える論理プログラミングの枠組みを使用する. それは一般論理プログラミング（normal logic programing；NLP）と呼ばれるものである. NLP は規則のボディに NAF が自由に現れることを許すプログラムである. しかし, NLP は不完全な知識を直接扱うことができない. 不完全な知識に対し, NLP は自動的に閉世界仮説（CWA）を適用する.

単調推論では持てる知識が増えたとき, 導かれる規則も増えるが, 否定的な知識を得たとき, 全体を矛盾としてしまうことになる. 帰納的な関係の生成において, 殊に今回のような正例と負例を扱う理論において適当ではない. 正例は目標概念の例を指すと考えたとき, 他の例は CWA により概念の例でないとされてしまうため, 負例の働きが正例に対する否定といった意味合いを持てなくなってしまう. CWA は, 正例よりその否定を完全に決定付けてしまい, 完全な分類を行ってしまう. 新しくヘッドの自由生成を考えている今回の場合, これは厳しい限定を受けることになる. この問題を解決することのできる理論として拡張論理プログラミング（extended logic programing;ELP）がある. この理論は NLP を論理否定‘ ¬ ’を扱えるように拡張したものである. ELP は, 三値に拡張した解集合を

持つ. プログラムの解集合において, ある例の正負のどちらの知識も持ち合わせていないとき, その例の真偽値は unknown である解釈される. そのため CWA の自動適用による負例の決定が避けられる.

現在の非単調推論によるデフォルト規則生成では, ある述語に対してその論理否定を持つものとの対が規則生成に用いられている. この限定を, 否定を内在する述語からでもデフォルト規則を生成可能にすることで更に柔軟な知識表現を実現することを目標とする. そのために否定を内在するソートを扱う理論を考察する.

第 2 章

非単調推論

2.1 非単調推論

ある時点で持ち合わせている知識の集合を A とし, この知識から導かれる結果が Th（A）という集合となったとする. 今, A に知識を加えていく. 新しく知識が加えられた集合の状態を B とする. B から導かれる事実を Th（B）とする. A よりも B の方が大きい. B が A を包含している. より多い知識からはより多くの事実が帰結できる, そのため,Th（A）よりも Th（B）の方が大きな集合を形成する. このような法則が成立する理論を単調推論と呼ぶ. 非単調推論はこのことが成立しない理論である.

この理論の重要な点として, 系における部分のみの考察が可能な点が上げられる. 単調推論における系（集合）の部分のみを捉えてみる. そして, そのときある事実が成立したとする. 今度は系全体に目を向けてみる. そうすると, 部分で成立した事実は系全体においても成立している. 図 2.1 参照. そのため, 一度証明された定理は未来において常に成立する. 知識の増加が定理を破壊することはない.

非単調推論は常に系全体を考慮しなければならない. 例を挙げる. 知識 A は

{ vehicle(x) $\wedge$ $\sim$airplane(x) $\rightarrow$ road(x) , airplane(x) $\rightarrow$ $\sim$road(x) , vehicle(mitubisi) }

を有するとする. この知識は乗り物であり飛行機でないならば, 道路上を移動することを 1 式目は表し, 2 式目は飛行機は道路上を移動しないことを記述している. 3 式目は mitubisi という実体は乗り物であることを示す. この知識からは

‘road(mitubisi)’

が導き出される. 次に A に

‘airplane(mitubisi)’

を追加し, この集合を B とする. この操作により先ほど導かれた

‘road(mitubisi)’
は導くことができなくなる. このとき閉世界仮説が働くことによりこの結果は発生する.

図 2.1: 単調推論の全系と部分の系の関係

図 2.2: 非単調推論の全系と部分の系の関係

先ほどのようにBで成立していたTh（B）がTh（A）より大きくなるとは限らない. 図2.2参照. このことは与えられる知識が不完全であるために起こる. もともと完全な知識の記述などできないので単調推論が考えられるシチュエーションは限定される.

よって非単調推論を使うことで不完全な知識から問題解決を試みる手法が考えられる. 逆に範囲を限定することにより有用な結果を導き出すことも可能になる.

例を考えてみる.「乗り物は道路を走行する」ものが多い. 一般的に乗り物とたずねれば自動車を想像し易い. そしてこのことを

$\forall x[\mathrm{vehicle}(x) \rightarrow \mathrm{road}(x)]$

と表すことにする. しかし例外的に飛行機や船舶といった道路上を移動しないものが存在する. よってこのままではxに飛行機に当たるものなどが入った場合に, 正しくない結果を導いてしまう. そこで, 成立させるために範囲の限定を考えてみる.

$\forall x[\mathrm{vehicle}(x) \wedge \sim\mathrm{airplane}(x) \rightarrow \mathrm{road}(x)]$

と表すことで乗り物に対しこの知識を利用できるようになる.

2.2 デフォルト推論

前章でのべた考え方はデフォルト推論という考え方に発展できる. 常識で考えれば「乗り物は道路を走行する」ものと表現する. このような常識という概念を規則生成や知識表現に使用することである.

デフォルト規則は常識を考慮する. このことにより未知の乗り物があったとき, その乗り物は道路を走るであろうと一応の結論を導き出せることになる. その未知の乗り物が飛行した事実がない限り, この結論は正しいものとして受け入れようという考えに立っている.

当然飛行する事がわかったときはこの導出した結論は破棄されるべきである. このような知識を表現するために導入された論理記号に様相演算子 M がある. そしてＭＰは「～Ｐが定理でないならばＭＰを定理としてよい」ことを表わす. この様相論理を使い先ほどの知識表現を行ってみる.

vehicle(x) ∧ M(∼airplane(x)) → road(x) ,
∀ x[airplane(x) → ∼road(x)],
vehicle(mitubisi)
この知識からは
'road(mitubisi)'
が導出される.
'airplane(mitubisi)'
という知識がないので
'M(airplane(x))'
が定理と使用されているため,
'mitubisi'
は飛ばないだろうと結論される.

2.3　正負例の反転・遅延

上述したように, 非単調推論を用いれば例外処理やデフォルト規則を生成することは可能となるが, 新たに正例と負例の扱いについての問題が発生する. 非単調推論では正例と負例の扱いがそのまま推論機構に影響を与える.

なぜならば, 生成器は, ある対立した知識を得た場合に, より一般的な方を正例と捉える. そのため, 負例の方がより一般的になった場合には, 正例と負例の反転が起こる. 生成器はこの反転により, 生成する規則を作り直せなければならない.

正例　E+={fly(a), fly(b)}
負例　E-={fly(c)}
背景知識　BG={bird(a), bird(b), bird(c)}

以上の知識からは

‘bird(x) ∧ Mfly(x) → fly(x)’

が生成される.

{∼fly(d), fly(e), bird(d), bird(e)}

が追加されることにより生成される規則は

‘bird(x) ∧ M∼fly(x) → ∼fly(x)’
となるべきである.

この場合, 正負の比に大差がないが, 一般的に多いものを常識的と考えることは自然である. この考え方に立つならば, 上知識から導かれる結論として ‘bird’ は ‘∼fly’ であって欲しい.

そしてこのことは閉世界仮説による「記述されていないことは偽である」という適用を避けなければならないことも意味する. 閉世界仮説のもとでは, デフォルト規則生成において, 例集合から一度正例を決定すると負例は正例に対する否定であるという限定を受けてしまう. この作用により, 負例は常に否定的な例として扱われ, 正例になることが許されず, 正負の反転は起こらないということになってしまう.

このため, 記述されていないことは無矛盾であるとして扱いたい. 正例が決定されたとき, その述語の否定を ‘否定である’ としない枠組みが必要となる. そのための理論として開世界特殊化（open world specialization；OWS）がある. この理論により負例を正例の否定という限定から開放できる.

第 3 章

規則生成

3.1 学習型拡張論理プログラミング

今回の研究において, 一般論理プログラミング (NLP) の拡張である, 拡張論理プログラミング (ELP) による学習を行う LELP を使用し, 規則生成を行う. そして従来の LELP に拡張を施す.

先ず, 一般規則と例外があるだけの場合を考える. このときの LELP のアルゴリズムを示す.

アルゴリズム１　LELP（E+, E-, BG, H）
入力：正例 E+, 負例 E-, 背景知識 BG
出力：規則 H := T 1 ∪ T2 ∪ T3

(1) E+, E-の個数を計算し多いものを正例とする.
(2) GenRules（E+, T, BG）

BG のもとで E+をカバーする規則 T を求める.

(3) OWS（T, E+, E-, BG, AB, T1）

OWS を用いて T を特殊化し, E-と BG に対して無矛盾となるようにデフォルト規則 T1 を求め, 同時に例外集合 AB も求める.

(4) Counter（E-, AB, T1, T2）

AB の下で E-をカバーする T2 を T1 と対応させた形で求める.

(5) Cancel（AB, BG, T3）

BG の下で AB を一般化し, デフォルト打ち消し規則 T3 を求める.

3.2 拡張論理プログラム

LELP では拡張論理プログラム（ELP）を使用する. ELP は次の形式の規則の集合として定義される.

$$\mathrm{L} \leftarrow \mathrm{L}_1 \ , \ldots \ , \mathrm{L}_m \ , \text{not } \mathrm{L}_{m+1}, \ldots \ , \text{not } \mathrm{L}_n$$

ここで $\mathrm{L}_i(0 \leq \mathrm{i} \leq \mathrm{n},\ \mathrm{n} \geq \mathrm{m})$ はリテラルであり, L をヘッド, $\leftarrow$ の右側をボディという. ELP では NAF を表わす not と論理否定の $\neg$ の 2 種類の否定が現れる.

常識の意味を簡単に捉えるなら L_1 から L_m が信じられており, L_{m+1} から L_n が信じられていないならば L が信じられることを表す.ELP の上記の形式の各規則を, デフォルトと同一視する.

$$\frac{\mathrm{L}_1 \wedge \ldots \wedge \mathrm{L}_\mathrm{m} : \overline{\mathrm{L}_{\mathrm{m+1}}}, \ldots \overline{\mathrm{L}_\mathrm{n}}}{\mathrm{L}}$$

ここでバーの付属している L は補リテラルを表す. このとき各解集合はデフォルト論理の各拡張に含まれるリテラル集合に一致する.

もしリテラル L が ELP P の全ての解集合に含まれるならば,L は P によって帰結するという. P のもとで L は成り立つという. LELP の入力として与える正例は正リテラルで表し, 負例は負リテラルで表す. 背景知識は ELP 形式の規則の集合である. ヘッドが正リテラルである規則を正の規則, 負のリテラルである規則を負の規則と呼ぶ.

BG を背景知識とし, E を正例と負例の集合, R を仮説とする. 全ての $\mathrm{e} \in \mathrm{E}$ に対して, $\mathrm{BG} \cup \mathrm{R}$ の下で e が導かれる（R は e をカバーする）ならば, R は BG と E に対して完全である. 任意の $\mathrm{e} \in \mathrm{E}$ に対して $\mathrm{BG} \cup \mathrm{R}$ の下で e の補リテラルが導かれない（R は e の補リテラルをカバーしない）のならば, R は BG と E に関して無矛盾である.

このとき正例と負例には同等の優先順位がつけられている. このことにより背景知識と例に対して無矛盾であるように, 正例と負例が生成された規則によりカバーされなければならない.

3.3　一般規則の生成

3.3.1　一般規則の生成条件

アルゴリズム１においてはGenRules（E,BG,T）で表される一般規則の生成が必要となる. これは一般規則Tが次の条件を満たすような極小の規則集合として生成され, 従来のILP手法を用いて実現される.

(１) TはBGとEに対して完全かつ無矛盾である.

(２) どんな例e∈Eに対してもT中にeと単一化可能なリテラルをヘッドに持った規則が存在する.

第２の条件はEを直接カバーする規則Tの生成を表し, １は一般化されない例が存在するときはそのままTとして加えられることを表わす.

3.3.2　一般規則の生成

GenRules（E,BG,T）アルゴリズムの例を示す.

アルゴリズム２　Genrules（E, BG, T）
入力：正例（または負例）E, 背景知識：BG
出力：一般規則T

(１) 背景知識BGから帰結される全ての基礎リテラル集合Mを生成する.
(２)EとMからRLGGを用いて規則の候補Rを生成する.
(３) リテラルの連鎖を用いてRに含まれる余分なリテラルを除去しR' を得る.
(４)R' のBGによるunfoldingによりTを得る.

このアルゴリズムの例を用いる上で, 帰結される基礎リテラル集合求めるため, BGに関数記号が含まれないという制限が設けられる.

RLGGはボトムアップ手法によって計算できる. しかし, RLGGで求めた規則のボディに多くの冗長なリテラルが含まれるので, この冗長なリテラルをリテラルの連鎖を用いて除去する. これは規則のリテラルの引数を基に, 有用なリテラルを抜き出す操作である.

次に動作例を示す.

E+ = { fly(1), fly(2), fly(3), fly(4) }
BG = { bird(1), bird(2), bird(3), bird(4), (bird(X) :- peng(X)), bird(c), peng(a), peng(b) }

BG から帰結される基礎リテラル集合を生成する.

M = { bird(1), bird(2), bird(3), bird(4), bird(a), bird(b), bird(c), peng(a), peng(b) }

fly(3) , fly(4) を基にした RLGG により

fly(x) :- peng(A), peng(B), bird(A), bird(B), bird(C), bird(D), ..., bird(X) ...

が生成される. この生成された規則にリテラルの連鎖を用いると

fly(X):-bird(X)

が得られる.

3.3.3 Unfolding

Unfolding は, 生成した規則のボディを短縮するために因子化とともに用いる. P をプログラム, C を規則とする. ここで C が（H：-β, A ,γ）の形であったとする.A は NAF でないリテラル, β, γ は（空かもしれない）リテラルの列であるとする. このとき, P による C の A に関する unfolding は, P を以下のプログラムに置換する.

（P から C を除去）$\cup$ {（H：-β, bd（D）, γ）θ | D $\in$ P かつ A が mgu θ で hd（D）と単一化可能 }

ここで規則 X に対して, hd（X）は規則 X のヘッドを表し, bd（X）は X のボディを表すものとする. 例を挙げる.

C = (ab1(X) :- bird(X) , peng(X))
P は (bird(X) :- peng(X)) を含む

このときの P による C の bird(X) に関する unfolding により

C = (ab1(X) :- pen(X))

で置き換えられる.

直感的には, 述語が‘ 何々ならば何々だ ’という読み方をしたとき, その結論に, 三段論法的に到達する過程が存在するが, その最初と最後の知識は結びついているわけである.

そこで, 最初の知識と最後の知識を結びなおし, 求めるべき知識の最短距離の述語を生成する. そして最短距離の知識み残し, その他のルートは削除するとみることができる. peng ならば bird なのだから,peng であれば ab1 であるといえる.

3.4 デフォルト規則生成

デフォルト規則生成において, 上述したとおり, 閉世界仮説を使用して規則生成を行うと負例の扱いが常に否定であると制限されてしまう. そのため, この適用を回避するために開世界特殊化を使用してデフォルト規則の生成を行う.

3.4.1 開世界特殊化

開世界特殊化により生成された規則はNAFを含むデフォルト規則の形式となる. 開世界特殊化において, 例外とは背景知識と一般規則から証明されるリテラルであり, その補リテラルが負例に含まれているようなものとなる. 開世界特殊化のアルゴリズムを示す.

アルゴリズム３　OWS（T, E+, E-, BG, AB, T’）
入力：規則 T , 正例 E+ , 負例 E- , 背景知識 BG
出力：デフォルト規則 T’ , 例外 AB

T 中の変数を含む各規則 Ci=（H：-B）に対して
Exc := {Lθ | L の補集合 $\in$ E-, BG $\cup$ T の基で Lθ が成立, BG $\cup$ (T より Ci を削除) は Lθ をカバーしない }；
If　Exc $\neq$ 0　then
If　B が NAF リテラル N を含む　then
AB := AB $\cup$ {Nθ | Lθ $\cup$ Exc}
Else　N:= abi（V1, ... ,Vn）
ここで {V1, ... ,Vn} は H 中の全変数であり
abi は BG 中には出現しない新しい述語

T’:= （T’ より Ci を削除）$\cup$ {（H：-B , ＮＡＦリテラル N）}
AB:= AB $\cup$ {Nθ | Lθ $\cup$ Exc}

このアルゴリズムから E-に含まれる L の補リテラルに対して, BG と T からは帰結されるが, BG と T から Ci を削除したものからは帰結されないような Lθ が Exc として集めら

れる. このことを言いかえると, Ci が加わることで Lθ が帰結されるのだから, Ci は BG と T から Lθ を求めるために必要な条件であると言えることとなる.

Genrules（E, BG, T）の (２) で Ci が生成されるとき, この Ci のヘッド H と E-に含まれる L の補リテラルとの間で導が可能となる. ここで非単調性の影響によりカバーされていた正例がカバーされなくなった場合には, カバーされなくなった正例を生成される規則 T' にそのまま追加する. 次にこのアルゴリズムの動作例を挙げる.

E+ = {fly(1), fly(2), fly(3), fly(4)}
BG = {bird(1), bird(2), bird(3), bird(4), (bird(X):-peng(X)), bird(c), peng(a), peng(b)}
T = {(fly(X) :- bird(X))}
E- = {-fly(a), -fly(b), -fly(c)}

ここで C １=(fly(X) :- bird(X)) とすると, C1 に対して

Ecx = {fly(a), fly(b), fly(c)}

である. そして N を ab1 と置くことにより

T' = {fly(X) :- bird(X), \+ab1(X)}
ab1={ab1(a), ab1(b), ab(c)}

3.4.2 反例を導く規則の生成

一般規則が生成され, そのとき例外として負例という扱いになって集められた集合に対し, 今度はその例外間に成立する規則を生成する. ここで生成される規則を, 反例を導く規則と呼ぶことにする. 本質的にはアルゴリズム２において正例が使用される場面において負例を使用することとなる. 次にそのアルゴリズムを示す.

アルゴリズム４　Counter（E, D, AB, T）
入力：例 E, デフォルト規則 D, 例外 AB
出力：規則 T

(1) R := {（H の補リテラル :- N）| Nθ は AB に含まれる
かつ (H :- B, \+N) は D に含まれる }
(2) T := R∪{e∈E | AB ∪ R によってカバーされない e の例が存在
する }

前述の背景知識, 例集合を使用すると, このアルゴリズムにより, 反例を導く規則は次のようになる.

-fly(X) :- ab1(x)

アルゴリズム４において AB が空の場合, R も空となる. その結果, 一般化は行われない. こうして AB が空の時は T=E が出力される. ここで T が D によってカバーされていない正例の導出を妨げないようにすると, 正例, 負例に対し一般規則が生成される. このことは正例と負例が同程度あるような場合や, どちらか一方を正例と決め付けてしまうことが困難であるような場合に効果を発揮する. 例として男女の比率などはどちらが正例であるか考えることは無意味である.

このような状況では形式上, 例は, 正例, 負例にわけ, どちらからも一般規則に相当する規則を生成することが考えられる.

3.4.3 デフォルト打ち消し規則の生成

先ほどは負例から反例を導く規則を生成した. 今度は負例と結び付けられる実体の間で成立する規則を生成することを考える.

OWS では例外の集合が AB として集められる. この集合内で成立する規則が存在することが考えられる. そこでこの集合に対し一般化を行い得られる規則を例外に関する規則と呼ぶ. この規則はデフォルト打ち消し規則として機能する.

OWS により集められた例外は述語記号 abi をもちいて記述されている. この述語記号 abi をもつリテラルをヘッドに含む規則が例外に関する規則である. Genrules アルゴリズムからも生成が可能である.

ここで例外は特殊な存在でなければならないという考えに立つと, 例外に関する規則があまりにも多くの帰結を起こすことは避けるべきである. そこで例外に関する規則が一般的でありすぎるかどうか判定を行う必要がでてくる. この場合与えられた AB 以外の帰結が起こらないように制限を設ける.

アルゴリズム５　Cancel（AB, BG, T）

入力：例外 AB, 背景知識 BG
出力：例外に関する規則 T

Genrules(AB, BG, T)
ここで BG ∪ T から帰結される abi の述語を持つ基礎リテラルの集合が AB の基礎例の集合と一致するように T を生成する

このアルゴリズムはアルゴリズム 2 での Genrules（E, BG, T）において E のかわりに A B を用い, 規則生成を行っている.

アルゴリズム 2 との相違点は, 生成規則に帰結するリテラルが AB に関するものである, という制限が加えられている点である. そして一般化がまったく行われない場合は, T は AB と一致する. そのためアルゴリズム 5 の出力は保証されることとなる. Cancel アルゴリズムの動作例を示す.

E+ = {fly(1), fly(2), fly(3), fly(4)}
BG = {bird(1), bird(2), bird(3), bird(4), (bird(X):-peng(X)), bird(c), peng(a), peng(b)}
例外の集合： AB ={ab1(a), ab1(b), ab1(c)}

Genrules の出力 T が

T = {ab1(X) :- bird(X)}

であったなら, BG∪T から帰結されるアトムは

ab1(1), ab1(2), ab1(3), ab1(4)

である. このアトムは例外と結びつくことはない. よって Cancel アルゴリズムの出力とはならない. そこで

T = {(ab1(X) :- peng(X)), peng(c)}

としてみると, 今度は, この規則と背景知識から帰結されるアトムは例外集合に一致する. そのため Cancel アルゴリズムの出力となる.

3.4.4 規則生成

上で述べたアルゴリズムを用いて規則生成を行う.

E = {fly(1), fly(2), fly(3), fly(4), fly(5), fly(6), -fly(a), -fly(b), -fly(c), -fly(d)}
BG = {bird(1), bird(2), bird(3), bird(4), bird(5), bird(c), (bird(X) :- peng(X)), peng(a), peng(b), peng(c), peng(d)}

上知識から以下の規則が生成される.

{(fly(X) :- bird(X), \+ab1(X)), (-fly(Y) :- ab1(Y), ab1(Z) :- peng(Z)}

以下に上知識での実体が存在する領域を表す.

図 3.1: 実体の存在する領域

第 4 章

規則生成器の拡張

4.1 推論規則の階層化

4.1.1 階層の概念

規則の階層化とは, たとえば, 動物には飛ぶものと飛ばないものが存在する. 一般的に動物は飛ばない. 例外として鳥類が飛ぶ. 今度は鳥類についてみてみると, 鳥類は一般的に飛ぶ. しかし, 例外としてペンギンやダチョウは飛ばない. このように規則の間に階層の関係が構築される.

4.1.2 階層内の構成

階層を構築する上で, 階層の名称として使用するものは, 生成された規則内の述語を反映させたものとする.

図 4.1: 階層の名称付け

4.1.3 規則生成におけるヘッド部の選択

この階層構造では, 生成される規則はヘッド部が, ある述語とその述語の否定であるという限定がされている. 現在この限定を取り除き, より柔軟な知識表現を行えるよう拡張している. この機能を視点の変化と考える. ここで視点とは, 規則を生成しようとする階層において, 例集合から生成規則のヘッド部に用いようとする述語が最適であると判断され, 規則が生成されるとする. そのときヘッド部に使用された述語とする.

4.2 非決定的規則

例外処理においてどちらが例外か判別しかねる状況がある. ひとつは考えることそのものが不自然であるような状況である. 例として「一般的に人は犯罪をしないものだ」という知識は, 例外として犯罪を犯す人間は人間でないという規則を生成してしまう.

もうひとつは正例と負例が近い比率で存在するような場合である. 例えば鳥が飛ぶことは多くの飛ぶ鳥を知っているという事実から導かれる. このことは次のような状況を作り出す. 飛ばない鳥が多く生息する地域では, 飛ぶ鳥と飛ばない鳥を半々に知っているような人物が存在し得る. この人物に「鳥は一般的に飛びますか？」と訪ねてもどちらとも言難いという答えが返ってくる可能性は十分にあり得る. ここで, この人物の知識を使って規則生成を行えば,「鳥は飛ぶか飛ばないか決めかねる」という規則が生成されてほしい.

もう少し詳しくすると, 例集合に正例と負例の比率が 50%付近であった場合, 正負のどちらの規則を生成することが正しいかの判断は困難になる. そのため判断してしまわず非決定性を導入することが考えられる. これには 2 つの考え方があり, 非決定的規則の生成と平行する例外を持つ規則の生成がある.

4.2.1 平行する例外を持つ規則の生成

平行する例外を持つ規則とは, 「哺乳類は一般的に飛ばないが例外としてコウモリは飛ぶ」ということと「鳥は一般的に飛ぶが例外としてペンギンは飛ばない」という 2 つの知識の間に成り立つ関係のことを指す. 要するに飛ぶことに関する例外と, 飛ばないことに関する例外が平行して存在することを表す. 例を挙げる.

fly-(X) :- mammal+(X), \+ab1(X)
ab1(X) :- bat+(X), mammal+(X)
fly+(X) :- bird+(X), \+ab2(X)
ab2(X) :- pengu+(X),bird+(X)
fly-(X) :- ab2(X)

以下に概念の図を示す.

図 4.2: 平行する例外を持つ規則の概念図

4.2.2 非決定的規則の生成

非決定性の考え方として次に挙げるものは非決定規則の生成である. 例としては「人間であり かつ 女でなければ男」という知識と「人間でありかつ 男でなければ女」を同時に存在させるような規則の生成を行う. この形式の規則は prolog では扱えない.NAF により推論がループに入ってしまうためである. しかし, 一般論理プログラムの安定モデル意味論 (stable model semantics) や拡張論理プログラムの解集合意味論 (answer set semantics) を使用することで, 「人間ならば女か男である」という解釈を持つことが可能となる. その例を示す.

male+(X) :- human+(X), \+female+(X)
female+(X) :- human+(X), \+male+(X)

概念図を示す.

図 4.3: 非決定的規則の概念図

4.2.3 非決定性規則の生成アルゴリズム

前節で説明したように非決定性の規則を生成するアルゴリズムを以下に示す.

アルゴリズム NML2(E+, E-, BG, R)
入力:正例 E+, 負例 E-, 背景知識 BG
出力:R = R1 ∪ R2 ∪ R3 ∪ R4

(1)E+と E-の比率より, 正の規則または負の規則を生成するか, 非決定的規則または平行する規則を生成するかを決定する
正の規則または負の規則を生成する場合は NML(E+, E-, BG, T1, T2) と同じとなる
非決定性の規則生成の状況と判断されたら以下の処理を行う
(2)E+と BG から, RLGG とリテラルの連鎖を用いて正の一般規則 R+を求める
(3)E-と BG から, RLGG とリテラルの連鎖を用いて負の一般規則 R-を求める
(4)OWS(R+, E+, E-, BG, AB1, R1)
(5)OWS(R-, E-, E+, BG, AB2, R2)
(6)R+と R-の中に互いに矛盾する節が存在するならば, R1, R2 を非決定規則に変換する. R3 = R4 = 0 として規則生成を終了する
(7)Cancel(AB1, BG, E+, R3)
(8)Cancel(AB2, BG, E-, R4)

このアルゴリズムを使用して非決定的規則の生成を行った例を示す

E = {fly+(1), fly+(2), fly+(3), fly+(4), fly-(a), fly-(b), fly-(c), fly-(d)}
BG = {bird+(1), bird+(2), bird+(3), bird+(4), bird+(a), bird+(b), bird+(c), bird+(d)}

この知識より次の規則が生成される

fly+(X) :- bird+(X), \+ab1(X)
fly-(Y) :- ab1(Y)
ab2(a), ab2(b), ab2(c), ab2(d)
fly-(Z) :- bird+(Z), \+ab2(Z)
fly+(W) :- ab2(W)
ab2(1), ab2(2), ab2(3), ab2(4)

しかしこのままでは生成された規則から 'ab1' と 'ab2' を除くと 'bird+(X)' で同じ式となってしまう.'bird+(5)' を追加すると両規則の 'ab1' と 'ab2' が証明されなくなってしまう. そのため互いに矛盾した規則となってしまう. そこで生成される規則を以下のように変更する.

fly+(X) :- bird+(X), \+ab1(X), \fly-(X)
fly-(Y) :- ab1(Y)
ab2(a), ab2(b), ab2(c), ab2(d)
fly-(Z) :- bird+(Z), \ab2(Z), \fly+(Z)
fly+(W) :- ab2(W)
ab2(1), ab2(2), ab2(3), ab2(4)

この例では正負の例の比率が 50%なので NML2(E+, E-, BG, R) アルゴリズムにおいて（２）以降の処理に進む.

4.3 述語自由生成

専門家と一般の人では常識も異なる. 例えば一般の人にビニールは燃えないゴミかと尋ねれば, 大体イエスという返事がかえって来ると思われる. しかし, もし同じ質問を専門

家に尋ねたのなら, 違う答えが返ってくる可能性がある. その人が住む町の焼却炉が高温燃焼のものならビニールは燃えるゴミとして扱って良い.

専門家は高温炉があるという前提ならビニールは燃えるゴミというだろう. しかし, 一般の人は, ビニールは燃えないゴミとみる.

もうひとつ例を挙げる. カンガルーは腹部に袋がある. この種を専門用語で有袋類という. じつはオオカミにも腹部に袋を持った種が存在する. そこでこの 2 種の動物をクラス分けしてみる.

専門的には有袋類が先に分類され, その種としてカンガルーとオオカミが存在することになる. しかし, 一般的に見れば, 動物がカンガルーとオオカミに分類され, その 2 つを見た結果, オオカミには有袋類が存在していた, とみると思われる.

まず専門家の見地から構成される階層を図に表す.

図 4.4: 専門家からみた動物の階層

この階層において, オオカミの上位概念が有袋類となる. 階層表現において, 抽象度は階層の上位の方が高いので, 有袋類の具体的な例としてオオカミが引用されていることをこの図は示していることになる. しかし, 今回の研究における階層概念は抽象度を包含的な意味で捉えている.

次に, 一般的な見地から構成される階層の図を表す.

先ほどの専門家の見地から構成される階層と違い, オオカミが有袋類よりも抽象的とされている.

このように見る者の知識や常識により構成される階層が変わる. そのため持っている知識の状態や見方により, 生成する規則にも変化が現れることは自然である.

この考え方が前節で述べた視点の変化である.

図 4.5: 一般的な見地から構成される動物の階層

4.4 否定的意味が内在する述語の生成器への組込み

4.4.1 強い否定における考察

強い否定や明白な否定によりコンピュテーションにおけるプログラムの枠組みは激変せざるを得ない. 理論体系を根本的なところより改革することとなる. 強い否定とは, 失敗による否定や, 明示されていないことによる否定とも取れる知識に対し, 直接的な言及をすることとなる. このことにより, ロジックプログラミングにおける経験的な知識からの推論の能力が, 不正確な述語に対し明確化する構造を備えることで強化することができる.

一般的にはロジックプログラミングはホーン節のセットから成る. ロジックプログラミングは古典論理の枠組みによる一階述語という文法的制限を受ける. ロジックプログラミングの証明には SLD - resolution を用いる. このプログラムでは a0←a1,a2,...an, という形式をしており, 古典論理より幾分柔軟である. しかし, このような節の論理的な振る舞いにおいて, 否定や補の状態をオペレートするための構造は存在しない. 古典的な解釈によるとこれらの方法は構造における文法の束縛力が強すぎることが言える. そこで SLD - resolution の間接的を使用することで prolog のメカニズムの説明に加え, この問題を明らかにする. prolog の証明手続きは自然演繹の方法をとる. このことは prolog のプログラマからみても自然であり考えやすい. そしてここを prolog 以降のロジックプログラミングは出発点とすべきである. そのため, 一般的なロジックにも触れる. ロジックプログラミングの意味論の発展をまず考察する. このとき出発の準備は一般的なロジック的な考察より行う.

そして, 強い否定を含む prolog のシステムの導入を行う. ここで以前からの枠組みは壊

さないよう新システムを構築する．構造として，３値, パーシャルロジックを用いる．この手法は top-down 型の評価を行う．そして negation as failure の操作を標準的な prolog と同じように備える．後にこのシステムの評価と適用されるべき問題の形式について説明する．

ここで従来の方法における含意の仕方は使用しない意味論での証明を行う．そのため, 最も単純な意味論から考察する．ロジックプログラミングの解釈として次のような考察がなされている．構造的なロジックのなかでホーン節に明確な解釈を与える方法とはどのようにすれば良いか．実際はホーン節と強い否定直感主義の否定, 古典的否定を使わないこととの間の差異を明確化することとなる．

ロジックプログラミングの否定の調査は, 暗黙の否定を含む情報の操作について論じられたものが大きい役割を果たす．失敗による否定を盛り込むものや, 閉世界仮説を関連付けるものが存在する．特に不一致による否定を使用した, 否定的情報を扱う理論は有用である．

不一致による否定は, 強い否定を局所的な概念として捉える．強い否定は偽を構成する．直接的にこの偽を表現することとなる．そのため, 直感主義やミニマルロジックでの古典的否定は間接的な否定表現であるといえる．そして証明において, 直感主義やミニマルロジックは間接的に証明不可能であることを否定として捉える．このことを考えると, 強い否定は AI のシステムにおける否定的情報を表現する上で, 自然な形で表すことができるといえる．

図 4.6: 古典論理の否定による間接的否定表現

図 4.7: 強い否定による直接的否定表現

AI の世界では強い否定について混乱した状況がみられる. prolog の中に表現される not を意味論的に強い否定として扱うことが見られる. しかしこれは弱い型の否定である. それか臨機応変に強い否定が必要とされたなら変化する. 例として, 次のような否定的なリテラルについて, 古典論理における否定の問題を避けるために ¬married(Jack, Jan) を not married(Jack, Jan) に変換する. この変換は強い否定の受け持てる範囲の正当化をする. このことはリテラルと対語の間の関係における矛盾を破壊する. これより導かれることは, 不一致の取り扱いについて, 一致しない知識においてはその関係を破棄することをベースとして推論を行うようにすることである.

そこで代わりの理論として提案するものが, 強い否定を意味する表現を明示的な否定的情報に付加することである. 否定的なファクトとして ∼married(Jack, Jan) をデータベースに追加する. もしこのデータベースから married(Jack, Jan) も推論できたなら, このデータベースは不一致が発生していることとなる.

4.4.2　強い否定により構成される集合の包含関係の考察

次に, 本研究において, 強い否定がどのような解釈をされるか, そして, その解釈の下で, どのように強い否定は扱われるかを述べる. そして強い否定を含む集合の包含関係の構築を行う.

公理として採用する関係を挙げる.

$\neg\neg \mathrm{p} \equiv \mathrm{p}$

$\mathrm{un\ p} \Rightarrow \neg \mathrm{p}$

$\mathrm{p} \cup \neg \mathrm{p} \equiv \mathrm{Universe}$

$\mathrm{p} \cap \neg \mathrm{p} \equiv \phi$

$\mathrm{p} \cap \mathrm{un\ p} \equiv \phi$

この公理群から構築される包含関係の図を以下に示す.

図 4.8: 強い否定を含む集合の包含関係の図

否定の包含関係において, 強い否定の 2 重否定がどのような包含関係で表されるか考察

する. この考察において２重否定には２つの解釈ができることが考えられる. 先ず, ここで否定的な関係ではない包含関係が構成する例を挙げる.

人物 A の友人に, 自分より「頭が良い」と感じている人物 B がいた. ここに新しい友人として人物 C と知り合った.B はここで知り合った C のことを「頭が良い人だ」と思った. そして, B はこの思ったことを A に「C は頭が良い人だ」と話し掛けた. このことを聞いた A は「C は B より頭が良いのだな」と思った.

次の考え方は, 構築される包含関係に否定的意味を持つ. しかし, その解釈が一意的に決定しないと思われる. 例を挙げる. 登場人物は上記の例と同じで, A, B, C とする.

A は, 友人 B のことを, 行うその行動が一般人からすると変わっていると感じていた. よって A は B のことを「B は変わり者だ」と認識している. 今, A と B に C が友人として加わる. B は C のことを「変わった人だ」と感じた. そしてそのことを A に「C は変わり者だ」と伝えた.

２つ例を挙げたのでこの例によって構築される包含関係の違いについて考えてみる. 先ず一つ目の例は包含関係を頭脳明晰度のような包含構成で構築しようとしている. 端的に言えば頭の良さによる包含関係である. 更に説明を加えれば, 人間全体の集合があり, その中に部分集合として小学生以上の年齢という集合があり, 次にその中に中学生以上の年齢という集合がある. 次は高校生というように, このような包含関係が, この A, B, C の３人の例で構成されると説明できる. B は A より頭が良い. よって B の属する集合は A の属する集合の部分集合である.

B の属する集合 $\subset$ A の属する集合

C は B よりも更に頭が良い. よって

C の属する集合 $\subset$ B の属する集合

次に二つ目の例で挙げた「変わり者」について考える. 今度の包含関係は常識度により構成される. A の属する集合は B の属する集合を包含しない. よって

B の属する集合 $\subset$ A の属する集合

は成立しないそして C は B から見れば変わっている者であるので

C の属する集合 $\subset$ B の属する集合

も成立しないここまでは問題なく包含関係を構成できる. では A の集合は C の集合をどのような形で包含しているだろうか. B は一般人からすれば変わっている. ならば, その者が下す判断は, 一般人とは変わった判断をするはずである. この解釈に従えば,C の属する集合は一般人の集合にのみ属し, B の属する集合には属していないはずである. 要するに C は変わり者ではない. しかし一般人である保証はない. 上記二つの例から構成される集合の包含関係の図を以下に示す.

図 4.9: 頭脳明晰度により構成される包含関係の図

図 4.10: 変わり者により構成される包含関係の図

意味付けの方法により一般人に戻るような解釈をすることは可能であると思われる.

この変わり者の例はいくつかの解釈ができる. B から見た C が変わり者であって A から見ると一般的に見えるという解釈以外に, C が B より更に変わり者の場合である. この解釈は頭脳明晰度と同じ包含関係となる. もう一つの解釈は「変わり者とは, 自分自身を一般基準として, 一般基準以外の人間を指す」とする解釈である. この解釈に従えば

un un p $\in$ p の属する集合

となるはずである. この点が変わり者の一つ目の解釈との相違点である. しかし, この解釈でも

un un p の属する集合　= p の属する集合

とはならないと思われる.

包含の不一致の問題において, 包含の内か外か決めるために無矛盾性の検証をすることも考えられる. ある集合の要素 a を, その集合の他の各要素について, 強い否定により二重否定を行い, その要素から無矛盾な関係が要素 a との間に成り立つか調べることを行う. ここで, 要素ごとに関連性がある場合は, その関連性を元に可逆性の検討を行うことが可能であると思われる. 例として, 生成可能な規則がある片寄りを示す場合や, 生成できる階層間の関係から逆に矛盾が導びかれることもあり得ることから, その事実より不可逆性の判断を行う場合などである.

4.4.3 否定的意味を持つ述語の組込み

規則生成においてある述語とその否定という形で構築を行ってきたが, ここで否定の役割を細分化することで, 生成される規則に更に具体性を増すよう拡張を行う.

接辞の否定 'un' は強い否定である. よって負例とされる例に包含される. そのため例集合より正例と負例をカウントする際に'un' の付属する例は負例としてカウントされる. しかし, 今回は意味的に結合することがシステム側で自動判別できないので, 負のリテラルが同じ述語の接辞否定のリテラルを包含するとして計算を行う. 例を付け加えると, 例えば

happy+(X)

という述語を考えたとき, その否定は

happy-(X)

であるが, 接辞否定'un' を含むリテラルの

happy-u(X)

はこの'happy-(X)' に包含される.

happy-u(X) $\subset$ happy-(X)

正の述語が正例として判断されたときは問題ないが, 負の述語が正例として判断されてしまった場合, 接辞の否定もそのまま正例として扱ってしまって良いかという問題は, 現在のところ負の述語による包含関係を維持するように規則生成を行うという制限により回避している. 最終的には包含関係に変化が生じると思われるので, この限定を取り除くことにより, 生成される規則は, 与えられた知識から多種の変化した規則を生成すると思われる.

次に今回採用する強い否定を含む規則生成器のアルゴリズムを示す.

If 正の述語数 > (負の述語+un の述語) 数 then
　正の述語を正例として計算
　E のラベル=‘+’;

Else
　負の述語を正例として計算
　E のラベル=‘-’;

begin:PROCEDURE
If Ab[n] = ϕ かつ E のラベル=‘+’

　If E(a) $\in$ E+ then
　　rule[n]=(E(a) :- BG(a));
　　Rule に rule[n] を追加;

　　If rule[n] の X 代入 = rule[n+1] の X 代入 then
　　Rule から rule[n+1] を削除;
　　対象がなくなるまで続行

　　If E(b) $\in$ E- かつ BG $\cup$ E+ $\models$ E(b) then
　　Ab=Ab $\cup$ E(b)

　If Rule[n] と同じ実体を持つものと単一化可能
　　E からその述語を削除;

　Rule[n]=Rule, Ab の単一化式

Else
　PROCEDURE を負の述語を正例として計算

If E(c) $\in$ Ab then

ab[n]=(E(c) :- BG(c));
AbRule に ab[n] を追加;

If ab[n] の X 代入 = ab[n+1] の X 代入 then
AbRule から ab[n+1] を削除;
対象がなくなるまで続行

end:PROCEDURE
If E+が空集合でない
PROCEDURE を続行

Else
end

第 5 章

アルゴリズムと評価

5.1 アルゴリズム

今回作成したプログラムのアルゴリズムの詳解をする.

5.1.1 多段例外処理アルゴリズム

まず, 正負例の反転のみを行うアルゴリズムを示す. 例集合には１種類の述語が含まれる. この述語の肯定と否定各々に実体が入り, 例集合は構成される. 今回の実装に持たせた知識を示す.

E = {kenko+(1), kenko+(2), kenko+(3), kenko+(4), kenko+(5), kenko-(6), kenko-(7)}

背景知識は実体の事実や形態を記述する. 背景知識は２種類の述語からなる.

BG = {sport(1), sport(2), sport(3), sport(4), sport(5), sport(6), sport(7), kakut(5), kakut(6), kakut(7)}

上例集合と背景知識と同じ実体をもつものをを結合する.

kenko+(1) :- sport(1), kenko+(2) :- sport(2), kenko+(3) :- sport(3), kenko+(4) :- sport(4), kenko+(5) :- sport(5), kenko+(6) :- sport(6), kenko+(7) :- sport(7), kenko+(5) :- kakut(5), kenko-(6) :- kakut(6), kenko-(7) :- kakut(7)

結合することで生成された組みのなかで, 背景知識の同じ物をカウントする. 上記の場合 'sport' に関する組みは７つであり, 'kakut' に関する組みは３つである. よって 'sport' の結

合している組みのヘッド部に使用されている述語の例の正負の記号が初期の規則の生成に使用される. 以下, この述語に対し, 規則は生成される. これにより生成される規則は

kenko+(1) :- sport(1), kenko+(2) :- sport(2), kenko+(3) :- sport(3), kenko+(4) :- sport(4)

である. そして, この生成した規則にリテラルの連鎖を用いて重複を消す. その結果

kenko+(X) :- sport(X)

が得られる. 例外の存在が確認された時点でこの規則には例外が考慮されるとして 'ab1' という例外があることを表すアトムが付加される

kenko+(X) :- sport(X), \+ab1

次にこの規則生成に使用されなかった例が例外として例外集合に集められる. 例外集合には

kenko+(5), kenko-(6), kenko-(7)

が集められる. そして背景知識からもこの知識と結び付けられる知識が例外処理を行うために集められる.

sport(5), sport(6), sport(7), kakut(5), kakut(6), kakut(7)

今度は先ほど使用した例の正負を逆転した述語について規則生成を行う. よって生成される規則は

kenko-(6) :- sport(6), kenko-(7) :- sport(7)

でありこれにリテラルの連鎖を使用し

kenko-(Y) :- sport(Y)

を得る. そしてまだ使用されていない例が存在するのでこの式に例外の存在するアトムが付け加えられる.

kenko-(Y) :- sport(Y), \+ab2

更にこの規則生成においても使用されなかった例が第 2 の例外処理として集められる

kenko+(5)

この例と結び付く背景知識を結合した時点で使用されていない例はなくなり, ここで規則生成は終了する. 生成された規則と例外集合をまとめると

kenko+(X) :- sport(X), \+ab1
ab1 = {kenko+(5), kenko-(6), kenko-(7)}
kenko-(Y) :- sport(Y), \+ab2
ab2 = {kenko+(5)}
kenko+(5) :- sport(5)

ここで生成された規則と階層の内容を説明すると, 「スポーツは一般的に健康に良い. 例外的に空手などの格闘技はスポーツであっても必ずしも健康のためになるとは限らない. しかし, 格闘技すべてがそうではなく, 格闘技の中にも型の競技などはその限りではない.」となる.

図 5.1: 多段例外処理アルゴリズム

5.1.2 視点の変化を考慮した例外処理アルゴリズム

次に前述のアルゴリズムに, 構成する階層を視点の変化により変化するよう拡張を施す.

今回のプログラムにおいては, 一般的な見地から階層をつくるとし, この場合, 多くの人が認めるものである階層を構築すべき, という点からより多く保持する知識について上位の階層を形成するようにアルゴリズムを構築した.

そして, このアルゴリズムは更に改良することで自然に述語の自由生成の機能を有することができる. 多くの人が支持する階層を生成することが述語の選択にも影響する. そのため, 必然的に階層構築にふさわしい述語と自由生成のための述語が結合される. それにより連鎖的に階層にあった規則の生成が発生する. その過程を以下に説明する.

システムに与えた例集合と背景知識を示す.

例 E = {kenko-(1), kenko-(2), kenko+(3), kenko+(4), kenko+(5), kenko+(6), glove+(7), kenko+(8), kenko+(9)}

背景知識 BG = {sport+(1), sport+(2), sport+(3), sport+(4), sport+(5), sport+(6), sport+(7),sport+(8), sport+(9), kakut+(1), kakut+(2), kakut+(3), kakut+(7),kata+(3), box+(7)}

背景知識BGの集合より, より多く含まれる述語を探し出す. ここでは'sport' である. よってこの知識と結び付く例の正負の数をカウントし, 多い方を初期生成規則の符合として選ぶ.

kenko-(1) :- sport+(1), kenko-(2) :- sport+(2), kenko+(3) :- sport+(3), kenko+(4) :- sport+(4), kenko+(5) :- sport+(5), kenko+(6) :- sport+(6), kenko+(7) :- sport+(7), kenko+(8) :- sport+(8), kenko+(9) :- sport+(9)

'sport' と結合する述語を探し出すことで上規則が生成される. ここで生成された規則のヘッド部には正のリテラルが負のリテラルより多いことから正の規則が生成されることとなる.

次に生成された規則の冗長性を取り除き, この時点で例外集合が空でないことが確認されるので, 冗長性を取り除いた規則に例外の集合の存在を表すアトムとして'ab1' を付加する. 例外集合もこの時生成する.

kenko+(X) :- sport+(X),\+ab1
ab1 = {kenko-(1), kenko-(2), kenko+(3), glove+(7)}

‘glove’ は述語の自由生成として後に規則生成に使用される．この時点では実体の追跡から’sport’ と結び付くことが確認できることで’sport’ の正負判別のカウントにより負例とされ, この述語に対する規則生成は, 例外として作用する．そのため ‘glove’ に対する正負の判別は’kenko-’ と同等の扱いを受ける．

今度は ab1 についてより多くの結合できる背景知識を探す. 結合可能な背景知識を計算するとここでは’kakut+’ が’kata+’ を上回る．

kenko-(1) :- sport+(1), kenko-(2) :- sport+(2), kenko+(3) :- sport+(3)

が生成される. 冗長性を取り除き, 例外が存在するか確認し, 更に存在するので例外集合 ab2 を付加し次の式と第 2 の例外集合を得る．

kenko-(Y) :- sport+(Y), \+ab2
ab2 = {kenko+(3), glove+(7)}

例外集合 ab2 に’kenko+’ と’glove+’ が同数含まれているため, 優先順位として, 現在規則のボディ部に使用している’sport’ との距離が短いものとして’kenko +’ を選ぶ. ここでの距離とは今, 対象としている述語がボディ部に結合するために何回規則生成を行ったかをカウントしたものである’kenko+’ は一回目の操作で’sport’ と結び付くが, ’glove+’ は一回目では’sport’ と’box’ に結び付く．よってより合致性が高いのは’kenko+’ であると判断する. 同列の時は前回生成規則に準ずるように規則生成を行う．

kenko+(3) :- sport(3)
kenko+(3) :- kata(3)

’sport’ を含む規則については前生成規則の例外であることから当然であるので実体のまま規則生成は終了する．しかし, ’kata’ を含む規則は次の例外処理を考慮するために実体を一般化する．

kenko+(3) :- sport(3)
kenko+(Z) :- kata(Z)

視点の変化により, 少数の人が関わるボクシングに対して, 大きな枠となる初期の規則を生成するよりも, 多くの人が関わるスポーツという枠により規則を生成した方が自然で

あることを, 規則生成の中に表現している. ボクシングをする者たちが大半を占め, ボクシングとはスポーツなのだ, という考えに立つことも考えられる. その場合は所有する知識にボクシングに関する知識が増加したときであるが, もちろん, そのような規則生成も可能である. ここでは, ボクシングをする者の見解にボクシングはスポーツではないという考え方は含まれていない. これは表現される知識以外の概念として考えられるためである.

図 5.2: 視点の変化を組み込んだアルゴリズム

5.1.3 否定の接辞 'un' を組み込んだ規則生成アルゴリズム

上記したように接辞の否定 'un' は強い否定であるので

happy-u(X) $\subset$ happy-(X)

に従い, 負例とされる例に包含することで規則生成を行う.

ここでは先ほど述べた視点の変化は組み込んでいないので, 例集合中の一般的述語を探し出すプロセスは入らない.

システムに与えた知識を以下に示す.

例 E = {kenko-u(1), kenko-u(2), kenko+(3), kenko-(4), kenko+(5), kenko+(6), kenko+(7), kenko+(8), kenko+(9)}

背景知識 BG = {sport+(1), sport+(2), sport+(3), sport+(4), sport+(5), sport+(6), sport+(7), sport+(8), sport+(9), kakut+(1), kakut+(2), kakut+(3), kakut+(4), kata+(3)}

上知識の例集合より正の述語と負の述語のカウントを行い, 多いものを正例にする. 今回は負の述語が負例となるように知識を与えている. よって, 第一段目の生成規則は '正の述語 $\Leftarrow$ 背景知識' となる. 生成される規則は

kenko+(3) :- sport+(3), kenko+(5) :- sport+(5), kenko+(6) :- sport+(6), kenko+(7) :- sport+(7), kenko+(8) :- sport+(8), kenko+(9) :- sport+(9), kenko+(3) :- kakuto+(3), kenko+(3) :-kata+(3)

生成された規則の中に現れる 'kenko+(3)' は背景知識のなかで 'sport+(3)','kakut+(3)', 'kata+(3)' と結合することができる. この背景知識から更に結合可能な例を探す. そうすると 'kenko-(X)' という負例と 'kenko+(3)' は関係することが判別される. そのため, 現状では負例を導く可能性のある規則は削除される. そして, 重複する規則を単一化して削除する. この時点で規則生成に使われなかった例を, 例外集合として集める.

ab1 ={ kenko-u(1), kenko-u(2), kenko+(3), kenko-(4)}

一般規則を生成する.

kenko+(X) :- sport+(X), $\setminus$ ab1(X)
ab1 ={ kenko-u(1), kenko-u(2), kenko+(3), kenko-(4)}

次に例外集合から規則をつくる. 強い否定を例外集合は含んでいるが, まず, 負例として扱える例に対し規則を生成する.

kenko-u(1) :- sport+(1), kenko-u(2) :- sport+(2), kenko-(4) :- sport+(4), kenko-u(1) :- kakut+(1), kenko-u(2) :- kakut(2), kenko-(4) :- kakut+(4)

先ほど生成した規則が 'sport+(X)' に関するものであったことから 'kenko-u(X) :- kakut+(X)' と 'kenko-(X) :- kakut+(X)' に関する規則は削除される. この考えは発展させることで視点の変化に結び付けることができるので, 前節で述べた視点の変化のアルゴリズムを組み込むことも可能である. 単一化を行って規則を一般化する.

kenko-(Y) :- kakut+(Y)

この規則に対する例外集合を生成する.

ab2 = {kenko+(3)}

この規則より無矛盾であることを確認し, 例外を考慮した規則を生成する.

kenko-(Y) :- kakut+(Y), ab2(Y)
ab2 = {kenko+(3)}

今, 上で「ここで負例として扱える例に対し規則を生成する」と述べた. 強い否定は, ある肯定の知識を一般的な否定より, より限定して作用することを, この表現で反映させる. この段階で 'kenko+(3)' に対する規則生成か, 'kenko-u(X)' による強い否定による規則生成となる. 今回は強い否定は負の述語により完全に包含できる解釈を採用するので, どちらの規則から生成しても階層に変化はない. この解釈に立たない場合, つまり完全な包含関係が形成できない場合は 'kenko-u(X)' を使った規則生成により 'kenko+(3)' の集合が破壊される可能性が発生する. では強い否定の規則生成を行い, 続けて単一化による一般化を行う.

kenko-u(Z) :- kakut+(Z), ab3(Z)
ab3 = { kenko+(3)}

例外集合 ab3 は上記解釈を採用するために付け加えられるものである. そして例外集合 ab2 の規則生成を行う.

kenko+(W) :- kauto(W)
kenko+(3) :- kata+(3)

ここで例集合に使われていない例が存在しなくなり規則生成を終了する. 生成された規則をまとめる.

kenko+(X) :- sport+(X), \+ab1(X)
ab1 ={ kenko-u(1), kenko-u(2), kenko+(3), kenko-(4)}
kenko-(Y) :- kakut+(Y), \+ab2(Y)
ab2 = {kenko+(3)}
kenko-u(Z) :- kakut+(Z), \+ab3(Z)
ab3 = { kenko+(3)}
kenko+(W) :- kauto+(W)
kenko+(3) :- kata+(3)

図 5.3: 強い否定を組み込んだアルゴリズム

5.2 評価

視点の変化による生成規則の変更は, 考え方そのものは有用であると思われる. 上のアルゴリズム例で示したように'sport' の示していることは'glove' の示していることより多くの実体と結び付き規則の生成を起こしている. 更にその生成される規則のヘッド部は 1 種類であるので一つの事柄についてたくさんの支持を得た規則であると解釈できる. 問題点として, 専門家のような特化した見地から階層を構築する場合の方法はまだ実現されていない. 専門家にとっては一般性以外に生物学的な方法や科学的な弁別の仕方もあるため, そのような基準に則った階層構築は大小関係という判断基準では記述できない.

強い否定の取り扱いは, 二重否定で元の集合に包含される解釈を採った. この解釈により, 古典的否定による二重否定の例から生成される規則が, 強い否定の例により生成される規則に破壊されることが避けられる結果を生む. しかし, 強い否定による二重否定は元の集合とは一致しない. そのため, 生成規則を逆にたどり, その結果導出される規則には, 常に成立する保証はなくなってしまっている. 更なる拡張はこの可逆性がどこまで成立するかを明白にすることである.

第 6 章

まとめ

本研究で行った点とシステムの拡張された点, 考察した点, そして, 今後の課題について述べる.

- 非単調推論に開世界特殊化をもちいて正負例の反転を行い規則生成を行った.
- 生成規則間に存在する階層関係を構築した.
- 規則生成の拡張として視点の変化を組み込み生成規則の柔軟性を向上した.
- システムがより一般性の高い述語を生成するために述語の自由生成を考察した.
- 否定の接頭辞'un' をシステムに組み込むための考察を行い, 強い否定に解釈を与え, 規則生成を行った.

まず, 第一のまとめより, 開世界特殊化により, 負例を否定である一意的状態から非決定的な状態に保持し, 後に正例とすることで例外処理による規則生成が可能であることを確認できた. 次に生成規則間に存在する階層関係を視点の変化と述語の自由生成により表現することができた. この表現により, 一般的な規則を生成することの意義がより明確になった. 否定が内在する述語をシステムに組み込むために, 強い否定の考察を行い, 強い否定の解釈の方法により, 構築される規則間の関係に変化があることを論じた. 強い否定の二重否定が, 元の集合と一致しないためにおこる非可逆性について有用な理論は考察段階である.

今後の課題を次に挙げる.

- 否定的意味を内在する述語の扱える範囲を 'un' のような接頭語のみでなく 'happy' と 'sad' のような対義語に対しても扱えるよう考察する.

- 背景知識と例集合の間に反対の関係の記述も可能であると思われる.
- 強い否定の二重否定により一致しない集合に対する考察を行う.

現在は, 否定的意味の内在する述語として強い否定のみを扱ってきた. 更に対義語に関しても適用できるよう拡張することが今後の課題である. 'un' のように明示的でない語の扱いになるので, その語中に否定が内在するかの判別を行うための枠組みが必要となる. そして本研究中では背景知識と例はシステムから見た第三者が決定している. しかし, この背景知識-例の関係を知識内から自動判別するような拡張は有用である. 規則のヘッド自体に導かれる内容の一般性の向上が見られることとなる. 二重否定が集合不一致を起こすことに対する解釈を与えて, 生成規則の可逆性をかのうにする定義を行うことができれば, いっそう柔軟な階層構築ができる.

参考文献

[1] Bain,M., Experimants in non-monotonic first-order induction, pp423–436, Academic press, London, 1992.

[2] Bain,M. and Muggleton,S., Non-monotonic reasoning, pp145–161, Academic press, London, 1992.

[3] Dimopoulos,Y. and Kakas, A. C., Leaning non-monotonic logic programs: leaning exceptions ,in:Proc.ECML-95, LNAI912,pp122–137, 1995.

[4] Muggleton,S., Inductive Logic Programing, Academic press, London, 1992.

[5] Gelfond, M. and Lifschitz, V., Classical negation in logic programs and disjunctive databases, New Generation Computing, pp365–385, 1991.

[6] G.Wagner, Vivid Logic:Knowledge-Based Reasoning with Tow Kind of Negation, Spring-Verlag, 1994.

[7] G.Wagner, Logic programming with strong negation and inexact predicates, Jounal of Logic Computation, 1991.

[8] Inoue,K.and Kudoh.Y, Leaning extended logic programing,Proc. of IJCAI-97,pp176–181,Morgan Kaufmann,1997.

[9] Leon Sterling, Logic Programing, the MIT press, 1995.

[10] Shan-Hwei Nienhuys-Cheng Ronald de Wolf, Foundations of Inductive Logic Programing, Springer, 1988.

[11] 井上克己, 中西博一, 失敗による否定を含む規則の自動生成, 人工知能学会資料, SIG-FAI-9601-25(08/02), pp180–187, 1996.

[12] 井上克己, 工藤嘉晃, 羽根田博正, デフォルト規則を含む拡張論理プログラムの学習, 人工知能学会誌, Vol.14,N0.3, pp437–445, 1999.

[13] 太田朗, 否定の意味, 大修館書店, 1980.

[14] 小野寛晰, 情報における論理, 日本評論社, 1994.

[15] 長尾真, 知識と推論, 岩波書店, 1988.

[16] 溝口文雄, 武田正之, 畝見達夫, 溝口理一郎, prolog とその応用, 総研出版, 1985.

[17] 森下真一, 知識と推論, 共立出版, 1994.

[18] 山崎進, 加藤弘之樹, 寺本光, 拡張論理プログラムに対する矛盾解消アブダクションの枠組, Vol.15,No4,pp719–726,2000.