社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－ICS－151 (20)
2008／3／7

帰納学習のための述語のメタ的性質とその自動生成に関する研究

石田 浩之† 中野 智文†† 犬塚 信博†

† 名古屋工業大学 情報工学専攻
†† 名古屋工業大学 情報基盤センター
E-mail: †raituti@phaser.elcom.nitech.ac.jp, ††{nakano.tomofumi,inuzuka}@nitech.ac.jp

あらまし 述語論理の枠組みで帰納的学習を行う帰納論理プログラミングは，背景知識を利用できるという利点の一方，仮説探索のコストが大きい．本研究は仮説の候補を意味的観点から絞り込む制御方法を提案する．仮説のトップダウン探索では，仮説を構成する節の本体に順に条件を追加する．このとき，条件が冗長である場合，あるいは条件が他の条件と矛盾してしまう場合があることに注目し，これらを検知する知識（述語のメタ的知識）を保持・利用する探索方法を与える．メタ的知識を形式化し，これが制御に用いるための原理を明らかにし，これを利用した学習システムを実装することで探索の効果を確認する．また，メタ的知識を自動的に生成するアルゴリズムを提案する．このアルゴリズムは外延的に述語の性質をテストし，冗長性と矛盾に関するメタ的性質を網羅的に生成する．
キーワード 帰納学習，帰納論理プログラミング，メタ的性質

Meta-knowledge for inductive learning and its generation

Hiroyuki ISHIDA†, Tomofumi NAKANO††, and Nobuhiro INUZUKA†

† Department of Computer Science and Engineering, Graduate School of Engineering, Nagoya Institute of Technology
†† Information Technology Center, Nagoya Institute of Technology
E-mail: †raituti@phaser.elcom.nitech.ac.jp, ††{nakano.tomofumi,inuzuka}@nitech.ac.jp

Abstract Inductive logic programming (ILP) is a framework of inductive learning based on logic programming. While ILP can use background knowledge, it makes the cost of search for hypothesis large. In this paper we propose a method to prune hypothesis using a kind of semantic knowledge. When an ILP system uses a top-down search, after it visits a clause (rule) it explore another clause by adding a condition. The added condition may be redundant with other conditions in the clause or the condition may causes the body of clause unsatisfied. We study to represent and use to treat the redundancy and unsatisfactory of conditions as meta-knowledge of predicates. In this paper we give a formalism of meta-knowledge and show to use it with an ILP algorithm. We also study a method to generate meta-knowledge automatically. The method generates meta-knowledge which controles redundancy and contradiction with respect to predicates by testing properties extensionally.
Key words inductive learning, inductive logic programming, meta-knowledge

1. はじめに

帰納論理プログラミング (ILP:Inductive Logic Programming) はデータを説明する仮説を導き，知識発見を行うための枠組みであり，分類学習やデータマイニングに広く応用されている [1]．分類学習はＩＬＰの代表的な応用であり，目標とする概念 (目標概念) の正しい事例 (正事例) と間違った事例 (負事例) が与えられ，目標概念の事例を説明するための他の概念の定義の集合 (背景知識) を使用して事例を説明する理論を得る．既に分かっている関連する知識を背景知識として利用できる点で他の手法に比べて優れる一方，反面仮説空間が非常に大きくコストが高い．

仮説探索の代表的方法は，単純な仮説から複雑な仮説へと探索していくトップダウン方式である．各仮説はその度，各事例が当てはまるかどうか検査される．そのため，トップダウン探索では仮説の候補数が探索時間に大きな影響を与えており，仮説の生成時に評価する必要のない仮説を排除する仮説生成制御法は重要である．

従来，入出力モードや型情報を用いた仮説生成制御法が知られている．入出力モードを用いた仮説生成制御は，入力引数は

それを持つリテラルを計算する際に具体化されていなければならないことを利用する制御である．例えば，1引数の述語 p において，その引数が入力となる場合，$g(A,B) \leftarrow p(C)$ は排除できる．型情報を用いた仮説生成制御は，引数の型に合った変数または定数でなければならないことを利用する制御である．2引数の述語 p の引数の型がそれぞれ *int*, *char* である場合，$p(A,B) \wedge p(B,C)$ は型の不一致によって排除できる．これ以外にも，意図する仮説以外を探索しなように文法上の制限を積極的に導入する方法が言語バイアスとして用いられてきた．ILP システムである Progol [6] や ILP に基づくデータマイニングシステム Warmr [4] [5] では独自のバイアス記述言語を用意している．

本研究は構文的ではなく意味的制約から仮説の生成を制限する方法に注目する．FOIL では恒の間の順序関係を発見する仕組みを導入し，順序に反する仮設を制限することを行ってきた [2]．本研究では意味的制限を一般的に検討する．

本論文では，第2章では ILP とそれにおける基礎知識および問題点について述べる．また，第3章では，まず，メタ的性質の具体例を示す．その後，メタ的性質の定義およびその場合の冗長と矛盾について論議し，自動生成法のアルゴリズムを提案する．第4章では，提案した自動生成法を実装し，メタ的性質を実際に生成可能か検証する．第5章では，まとめと今後の課題について述べる．

2. 準　　備

2.1 論理と論理プログラミング

$\forall(h \leftarrow b_1, b, \ldots, b_n)$ の形式を節（clause）という．ここで，$h, b_1, b_2, \ldots, b_n$ は述語と項で構成されたリテラルであり，$\forall(P)$ は P の中にある変数が全て全称限定されていることを意味する．節であることがわかる場合は，$\forall$ を略して $h \leftarrow b_1, b, \ldots, b_n$ と書く．このとき，h は節の頭部，$b_1, b_2, \cdots, b_n$ は節の本体と呼ばれる．

述語論理においてある理論を考えるとき，そこで用いる定数記号，関数記号，述語記号を一つの記号集合 $\mathcal{L}$ に含まれていると考える．$\mathcal{L}$ の記号に様々な意味を割り当てたものを**解釈**と呼ぶ．

即ち，対象領域 U に対する $\mathcal{L}$ の解釈 $\mathcal{I}$ は $\mathcal{L}$ の各記号に次のとおり割当てることである：

（1） 定数記号に U のある要素を与える．

（2） n 引数の関数記号 f にある関数 $\tilde{f} : D^n \rightarrow D$ を与える．

（3） n 引数の述語記号 p にある述語 $\tilde{p} : D^n \rightarrow \{\text{true}, \text{false}\}$ を与える．

解釈 $\mathcal{I}$ に対し，$|\mathcal{I}|$ は $\mathcal{I}$ の対象領域を表す．知識ベース設計者などのユーザーが意図した解釈を特に**意図的解釈**と呼ぶ．解釈 $\mathcal{I}$ で理論 Δ Δ が正しいとき $\mathcal{I} \models \Delta$ と書く．また，理論 Δ について $\mathcal{I} \models \Delta$ となる任意の解釈 $\mathcal{I}$ で，理論 Γ について $\mathcal{I} \models \Gamma$ となるとき，$\Delta \models \Gamma$ と書く．

分類学習では，いくつかの事実を説明する論理式を求めたいことが多い．ある論理式が事実（事例）を説明できるとき，分類学習の文脈では，論理式が事例を**被覆**(coverage) するという．即ち，ある理論（＝論理式集合）$\mathcal{B}$，論理式 e, h に対し，

$$\mathcal{B} \cup \{h\} \models e$$

となるとき，h が $\mathcal{B}$ に関して e を**被覆する**(cover) という．

```
入力 ： 正事例集合 P，負事例集合 N，背景知識 B
出力 ： 正事例を全て被覆し，負事例を全く被覆しない理論 T
 1. T ← ∅
 2. P' ← P
 3. while P' ≠ ∅ do
 4.    N' ← N
 5.    clause ← 空の節「目標 ⇐ 」
 6.    while N' ≠ ∅ do
 7.       C ← clause に付加可能なリテラルの集合
 8.       C の中で最も適したリテラル L を clause に付加
 9.       N' ← N' − {e ∈ N' | B ∪ {clause} ⊭ {e}}
10.    T ← T ∪ clause
11.    P' ← P' − {e ∈ P' | B ∪ {clause} ⊨ {e}}
```

図1　FOIL のアルゴリズム

2.2 帰納論理プログラミング

ILP では，事実を説明 (被覆) するために関連する対象の知識などの理論を用いる．これを**背景知識**(background knowledge) といい，$\mathcal{B}$ と表す．説明される対象は通常，同じ述語の基礎原子文の形式をとり，**事例**という．事例に使われる述語のことを**目標述語**(target predicate) または**目標概念**(target concept) という．

帰納論理プログラムの論理設定

帰納論理プログラミングでは，正事例集合 $\mathcal{P}$ および負事例集合 $\mathcal{N}$，背景知識 $\mathcal{B}$ に対し，次の二つが前提となる．

1. 背景知識 $\mathcal{B}$ が被覆しない正事例集合 $\mathcal{P}$ の要素，正事例 p が存在する (事前不十分性)
2. 背景知識 $\mathcal{B}$ および負事例集合 $\mathcal{N}$ はどんな理論も判意しない (事前無矛盾性)

この二つのどちらも成り立つ場合に，次を満たす理論 $\mathcal{T}$ を見つけることが目的となる．

1. 背景知識 $\mathcal{B}$ および理論 $\mathcal{T}$ によって正事例集合 $\mathcal{P}$ の要素が全て被覆される (事後十分性)
2. 背景知識 $\mathcal{B}$ および理論 $\mathcal{T}$，負事例集合 $\mathcal{N}$ はどんな理論も判意しない (事後無矛盾性)

トップダウン ILP の例として FOIL [2] [3] を図1に示す..

FOIL のアルゴリズム

図1のアルゴリズムは二重のループでできている．外側のループでは事後十分性を理論 $\mathcal{T}$ が満たすまで節を加えている．また，内側のループでは事後無矛盾性を満たす節 *clause* を生成している．節 *clause* の生成法は，まず，背景知識から *clause* に付加可能なリテラルの内，適切なリテラルを選択し，そのリテラルを *clause* に加える．

「付加可能」なリテラルの個数によって，探索空間の大きさ

が決まる．これは構文的に付加できるだけでなく，何らかの制限によって大きさを制御する必要がある．「適切」さについては情報量の観点などから基準が与えられる．

仮説の形式を制御することを言語バイアスという．従来の言語バイアスには入出力モードや型情報による制御が挙げられる．入出力モードによる制御は，関数の考え方に由来する．関数は入力を与えられなければ出力を出すことができない．述語で構成されたリテラルについて書く引数に入力／出力の役割を決めておくことで，入力引数はそのリテラルが用いられる前に具体化されていなければならないことを制約することができる．型情報による制御は，C 言語や Pascal などで使われる型の概念と同じである．つまり，述語の引数はそれに与えられた型の変数または定数が与えられなければならないという制限である．

3. メタ的性質とその定式化

3.1 意味的性質から排除可能な仮説

本章では意味的性質が仮設を制限できることを例にとって示す．以下の二つの述語が背景知識にある場合を考えてみる．

- $\mathrm{inc}(A,B)$:自然数 A,B に対し $B=A+1$
- $\mathrm{inc}(A,B)$:自然数 A,B に対し $B=A-1$

このとき，$\mathrm{inc}(A,B)\wedge\mathrm{inc}(A,B)$ は充足不能であり，この2つのリテラルを含む節は事例を一つも被覆することがない．即ち，リテラル $\mathrm{inc}(A,B)$ とリテラル $\mathrm{inc}(A,B)$ は意味的に矛盾する．また，$A+1=B$ と $B-1=A$ は同じことを意味し，したがって $\mathrm{inc}(A,B)\wedge\mathrm{inc}(B,A)$ は，単に $\mathrm{inc}(A,B)$ 以上の意味を持たない．すなわち，$\mathrm{inc}(A,B)$ と $\mathrm{inc}(A,B)\wedge\mathrm{inc}(B,A)$ は被覆する事例が変化がなく，このような場合，新たに追加した $\mathrm{inc}(B,A)$ は冗長である．さらに，$\mathrm{inc}(A,B)$ から A は B よりも数として小さいことが分かる．つまり，A と B がとる値において順序関係が成り立ち，推移性および反対称性を持つ．したがって，$\mathrm{inc}(A,B)\wedge\mathrm{inc}(B,C)$ であれば，ここから A と C の間に順序が成り立ち，$\mathrm{inc}(A,C)$ とは矛盾する．$\mathrm{inc}(A,B)$ と $\mathrm{inc}(A,C)$ ならば $B=C$ である ($\mathrm{inc}(A,B)$ は一意性を持つ) も制御に利用できる．ここで挙げたものがメタ的性質の一例である．

3.2 メタ的性質の定式化

ここではメタ的性質の定義の定式化を行う．まず，いくつかの定義を準備する．述語 inc を考えたとき，数の間の順序関係を考える必要があったように，メタ的性質を議論するとき，単に知識ベースに含まれる述語だけでなく，関連する述語を用意する必要がある．そこで知識ベース **DB** に対して，

- **DB** で使われている定数，関数，述語記号，および
- **DB** から暗に示される意味を表わすための述語記号

の集合を **DB** の関連記号集合と呼び，$\mathcal{L}_{\mathbf{DB}}$ と書く．

このとき，次の通りメタ的性質を定義する．

定義 1 (メタ的性質) **DB** を背景知識および目標概念のデータを含む知識ベース，$\mathcal{L}_{\mathbf{DB}}$ を **DB** の関連記号集合，$\mathcal{I}$ を $\mathcal{L}_{\mathbf{DB}}$ の意図的解釈とする．このとき $\mathcal{I}\models\mu$ を満たす関数記号および定数記号を使用していない論理式 μ を，**DB** における**メタ的性質**と呼ぶ．μ の頭部が目標概念でない場合を特に**真のメタ的性質**と呼ぶ．

次から示す表記法および定理を用いて定理 1 を証明する，

太字 $\boldsymbol{x}, \boldsymbol{x_1}$ 等で，変数の組を表わすとする．また，リテラル l や論理式 F に $\boldsymbol{x}$ が含まれていることを強調して $l(\boldsymbol{x})$，$F(\boldsymbol{x})$ と書く．$\boldsymbol{x}, \boldsymbol{x_1}$ 等の各変数を異なる新しい定数に置き換えた定数の組を，$\boldsymbol{c_x}, \boldsymbol{c_{x_1}}$ 等と書く．リテラル $l(\boldsymbol{x})$ や論理式 $F(\boldsymbol{x})$ の $\boldsymbol{x}$を新しい別々の定数に置換えたものを $l(\boldsymbol{c_x})$，$F(\boldsymbol{c_x})$ と書く．このとき次のことが成立つ．

定理 1 (メタ的性質の拡張) $\mathcal{M}$ を知識ベース **DB** のメタ的性質の集合，$l_1(\boldsymbol{x_1}),\cdots,l_n(\boldsymbol{x_n}),l'(\boldsymbol{x'})$ を定数および関数記号を含まないリテラルとする．このとき，

$$\mathcal{M}\cup\{l_1(\boldsymbol{c_{x_1}}),\cdots,l_n(\boldsymbol{c_{x_n}})\}\vdash l'(\boldsymbol{c_{x'}}) \qquad (1)$$

が成立つならば，$l_1(\boldsymbol{x_1})\wedge\cdots\wedge l_n(\boldsymbol{x_n})\rightarrow l'(\boldsymbol{x'})$ はメタ的性質である．

証明 式 (1) に演繹定理を適用して，次式が得られる．

$$\mathcal{M}\vdash l_1(\boldsymbol{c_{x_1}})\wedge\cdots\wedge l_n(\boldsymbol{c_{x_n}})\rightarrow l'(\boldsymbol{c_{x'}})$$

このとき，$\boldsymbol{c_{x_1}},\ldots,\boldsymbol{c_{x_n}},\boldsymbol{c_{x'}}$ は自由変数とみなせることに注目すると次の式が得られる ($\forall$ の導入規則と考えてよい)

$$\mathcal{M}\vdash\forall\boldsymbol{X}(l_1(\boldsymbol{x_1})\wedge\cdots\wedge l_n(\boldsymbol{x_n})\rightarrow l'(\boldsymbol{x'}))$$

ここで，$\boldsymbol{X}$ は $\boldsymbol{x_1},\cdots,\boldsymbol{x_n},\boldsymbol{x'}$ を併せたものである．
$\mathcal{M}$ はメタ的性質であり，即ち，$\mathcal{I}\models\mathcal{M}$ であるので，

$$\mathcal{I}\models\forall\boldsymbol{X}(l_1(\boldsymbol{x_1})\wedge\cdots\wedge l_n(\boldsymbol{x_n})\rightarrow l'(\boldsymbol{x'}))$$

である．よって，定理が示された．

すると次の2つはここから直ちに分かる．

系 1 (冗長) $F(\boldsymbol{x})$ を連言形式の論理式，$l(\boldsymbol{x'})$ を$\boldsymbol{x}$の一部の変数$\boldsymbol{x'}$をもつリテラル，n を $\boldsymbol{x}$ の変数の数とする．このとき，$F(\boldsymbol{x})\rightarrow l(\boldsymbol{x'})$ がメタ的性質ならば次が成立つ．

$$\{\boldsymbol{x}\in|\mathcal{I}|^n\mid\mathcal{I}\models F(\boldsymbol{x})\}=\{\boldsymbol{x}\in|\mathcal{I}|^n\mid\mathcal{I}\models(F\wedge l)(\boldsymbol{x})\}$$

系 2 (矛盾) $F(\boldsymbol{x})$ を連言形式の論理式，$l(\boldsymbol{x'})$ を$\boldsymbol{x}$の一部の変数$\boldsymbol{x'}$をもつリテラル，n を $\boldsymbol{x}$ の変数の数とする．このとき，$F'(\boldsymbol{x'})\wedge l(\boldsymbol{x''})\rightarrow\square$ がメタ的性質ならば次が成立つ．

$$\{\boldsymbol{x}\in|\mathcal{I}|^n\mid\mathcal{I}\models(F\wedge l)(\boldsymbol{x})\}=\emptyset$$

定理 1，1，2 を利用しての候補の制御法は図 2 になる．

$\mathcal{M}$ を知識ベース DB のメタ的性質の合集とする．このとき，節 $c = t(x) \leftarrow l_1(x_1) \wedge \cdots \wedge l_n(x_n)$ に対して，次のうち（少なくとも）1つが成立つとき，節 $c' = t(x) \leftarrow l_1(x_1) \wedge \cdots \wedge l_n(x_n) \wedge l'(x')$ を探索しない．

（1） $\mathcal{M} \cup \{l_1(c_{x_1}), \cdots, l_n(c_{x_n})\} \vdash l'(c_{x'})$ かつ x' の全ての要素が $x_1, \cdots, x_n$ のどれかの要素である．

（2） $\mathcal{M} \cup \{l_1(c_{x_1}), \cdots, l_n(c_{x_n}), l'(c_{x'})\} \vdash \square$

図 2 メタ的性質を用いた仮説生成制御法

入力 ： 正事例集合 $\mathcal{P}$，負事例集合 $\mathcal{N}$，背景知識 $\mathcal{B}$
出力 ： 正事例を全て被覆し，負事例を全く被覆しない理論 $\mathcal{T}$

1. $\mathcal{T} \leftarrow \emptyset$
2. $\mathcal{P}' \leftarrow \mathcal{P}$
3. while $\mathcal{P}' \neq \emptyset$ do
4. $\quad \mathcal{N}' \leftarrow \mathcal{N}$
5. $\quad clause \leftarrow$ 空の節「目標 $\Leftarrow$」
6. $\quad$ while $\mathcal{N}' \neq \emptyset$ do
7. $\quad\quad \mathcal{C} \leftarrow clause$ に付加可能なリテラルの集合
8. $\quad\quad$ for each $L \in \mathcal{C}$
9. $\quad\quad\quad$ if L が図 2 で加えないと判断される then
10. $\quad\quad\quad\quad L$ を $\mathcal{C}$ から排除
11. $\quad\quad \mathcal{C}$ の中で最も適したリテラル L を $clause$ に付加
12. $\quad\quad \mathcal{N}' \leftarrow \mathcal{N}' - \{e \in \mathcal{N}' | \mathcal{B} \cup \{clause\} \not\models \{e\}\}$
13. $\quad \mathcal{T} \leftarrow \mathcal{T} \cup clause$
14. $\quad \mathcal{P}' \leftarrow \mathcal{P}' - \{e \in \mathcal{P}' | \mathcal{B} \cup \{clause\} \models \{e\}\}$

]

図 3 メタ的性質を用いた FOIL のアルゴリズム

4. メタ的性質を利用した仮説生成制御を持つ学習システム

FOIL アルゴリズムもとに前章で示した仮説生成制御法を組み込んだ学習アルゴリズムを図 3 に示す．図中の下線部が新たに加わった部分である．表 1 に，述語に見られる主なメタ的性質を示す．従来から用いられている型による制御もメタ的性質として扱うことが可能である．これらのメタ的性質を利用した帰納実験を行った．ただし，入出力モードによる仮説制御はこの方法で直ちに扱うことができない，そこでこれらは従来の方法で組み込んだ．

要素 A がリスト B に含まれるかどうかに関する述語 member(A, B) を目標概念としてメタ的性質による仮説制御の効果を確認する帰納実験を行った．このとき，背景知識には $A = [B|C]$ であることを表わす述語 component(A, B, C) を用いた．このときに使用するメタ的性質は以下のものである．

- $member(A, \_) \rightarrow element(A)$
- $member(\_, A) \rightarrow list(A)$
- $component(A, \_, \_) \rightarrow list(A)$
- $component(\_, A, \_) \rightarrow element(A)$
- $component(\_, \_, A) \rightarrow list(A)$
- $element(A) \wedge list(A) \rightarrow \square$
- $component(A, \_, B) \rightarrow trans_{com}(A, B)$

表 1 各性質とメタ的性質の対応

性質名	説明	使用するメタ的性質
型	$p(X, \cdots)$ の変数 X の型は A	$p(X, \cdots) \rightarrow typeA(X)$ $typeA(X) \wedge typeB(X) \rightarrow \square$
対称性	$p(X, Y)$ は対称性を持つ	$p(X, Y) \rightarrow p(Y, X)$
2述語の対称性	$p(X, Y)$ は $q(Y, X)$ と対称性を持つ	$p(X, Y) \rightarrow q(Y, X)$
反対称性	$p(X, Y)$ は反対称性を持つ	$p(X, Y) \wedge p(X, Y) \rightarrow \square$
推移性	$p(X, Y, \cdots)$ の $\langle X, Y \rangle$ は推移する	$p(X, Y, \cdots) \rightarrow trans(X, Y)$ $trans_p(X, Y) \wedge trans_p(Y, Z) \rightarrow trans_p(X, Z)$
一意性	$p(X, Y)$ は X を入力引数，Y を出力引数としたとき，一意性を持つ	$p(X, Y) \rightarrow uni_p(X, Y)$ $uni_p(X, Y) \wedge uni_p(X, Z) \wedge Y \neq Z \rightarrow \square$
反射性	$p(X, Y)$ は反射性を持つ	$p(X, X)$
非反射性	$p(X, Y)$ は非反射性を持つ	$p(X, X) \rightarrow \square$

表 2 メタ的性質を用いて $member(A, B)$ を帰納したときの仮説数

ループ	従来法の探索仮説数	メタ的性質を利用時の探索仮説数	
		型のみを利用	全性質を利用
1回目	4	4	2
2 回目	4	4	2
3回目	21	21	6

- $trans_{com}(A, B) \wedge trans_{com}(B, A) \rightarrow \square$
- $trans_{com}(A, B) \wedge trans_{com}(B, C) \rightarrow trans_{com}(A, C)$
- $component(A, B, C) \rightarrow uni_{com}(A, B, C)$
- $uni_{com}(A, B, \_) \wedge uni_{com}(A, C, \_) \wedge B \neq C \rightarrow \square$
- $uni_{com}(A, \_, B) \wedge uni_{com}(A, \_, C) \wedge B \neq C \rightarrow \square$

実験結果を表 2 に示す．ループ回数は，トップダウンアルゴリズムの内側のループでふさわしいリテラルを選ぶためにリテラルを探索する回数である．型に関する性質のみを利用したとき，従来の探索と同じ数の仮説を探索した．これ以外のメタ的性質を利用したとき，多くの仮説をテストすることなく棄却している．メタ的性質をテストするためにコストを要するため，直ちに効率の改善を意味しないが，事例数が大きなときに効果が期待できる．

5. メタ的性質の自動生成法

メタ的性質は，述語の意味内容を熟知していなければ与えることができない．通常背景知識は既に知られた知識であり，このことは必ずしも制限に当たらないが，メタ的性質を与えずに自動的に生成することができれば，適用範囲が広がる．そこでここでは，外延データーベースにおける自動生成法のアルゴリズムを提案する．メタ的性質を生成する方法として，述語の外延を求め，それをもとに外延的に帰結できる性質を求めること

meta_generator
入力 ： 述語集合 $\mathcal{P}$, $\mathcal{P}$ の外延データーベース **DB**
出力 ： メタ的性質 $\mathcal{M}$
1. $\mathcal{M} \leftarrow \emptyset$
2. for each $p/n \in \mathcal{P}$
3. $\quad \mathcal{M} \leftarrow \mathcal{M} \cup sub\_generator(p(X_1, \cdots, X_n), \mathcal{P}, \mathbf{DB})$

sub_generator
入力 ： 連言形式の論理式 F, 述語記号集合 $\mathcal{P}$,
$\quad$ $\mathcal{P}$ の外延データーベース DB
出力 ： 節の本体に F が必ず現れるメタ的性質 $\mathcal{M}$
1. $\mathcal{M} \leftarrow \emptyset$
2. for each $p/n \in P$
3. $\quad$ 次の条件を満たすリテラルの集合 $\mathcal{L}$ を生成
$\quad$ •p/n を使っている
$\quad$ •F と少なくとも一つ変数を共有している
4. $\quad$ for each $l \in \mathcal{L}$
5. $\quad\quad$ if $Var\_Set(l) \subseteq Var\_Set(F)$ then
6. $\quad\quad\quad$ if $\{x|DB \models F(x)\} = \{x|DB \models F \wedge l(x)\}$then
7. $\quad\quad\quad\quad \mathcal{M} \leftarrow \mathcal{M} \cup \{"l \leftarrow F"\}$
8. $\quad\quad\quad$ if $\{x|DB \models F \wedge l(x)\} = \emptyset$ then
9. $\quad\quad\quad\quad \mathcal{M} \leftarrow \mathcal{M} \cup \{"\square \leftarrow F \wedge l"\}$
10. $\quad\quad$ else
11. $\quad\quad\quad NewF \leftarrow "F \wedge l"$
12. $\quad\quad\quad \mathcal{M} \leftarrow \mathcal{M} \cup sub\_generator(NewF, \mathcal{P}, DB)$

図 4 自動生成法のアルゴリズム

ができる．また，述語の内包定義が与えられている場合は，その定義から演繹的にメタ的性質を求めることも考えられる．ここでは外延的な自動生成のアルゴリズムを検討する．図 4 にアルゴリズムを示す．

アルゴリズムは二つの手続き *meta_generator*, *sub_generator* からなる．アルゴリズム中，$Var\_Set(F)$ は論理式 F の変数の集合を返す関数，$Var\_List(F)$ は論理式 F の変数のリストを返す関数である．

これらのアルゴリズムはあるリテラルの連言から外延的に帰結できることを網羅的に探索する．*meta_generator* は各述語から作られる 1 つのリテラルのみを条件に，そこから帰結できる条件，矛盾する条件を *sub_generator* を呼び出すことで求める．*sub_generator* の 6 行目と 7 行目のテストがこの 2 つに相当する．テストするための条件を 2，3 行目で網羅的に求め，これらのテストにかかった条件から，冗長性に関するメタ的性質と，矛盾に関する性質をそれぞれ生成する．どちらのテストにも掛からない場合は，この条件を更に前提条件に加えて 1 2 行目で再帰的に *sub_generator* を用いてメタ的性質を求める．このアルゴリズムは，停止条件が示されておらず，適度な深さで停止させる必要がある．

6. メタ的性質の自動生成に関する実験

提案したメタ的性質の自動生成アルゴリズムを Prolog で実装した．本章ではこれを用いてメタ的性質の自動生成に関する実験を行う．メタ的性質の構文そのものを制限する言語バイアスとして型情報を用いた．

実験では背景知識を外延的に与えた．実験に使用した背景知識とその外延データベースを以下に示す．

対象領域： $\{1, 2, 3, 4, 5, a, b, c, d, e\}$
外延データーベース：
$add(A, B, C)$：意味「$A + B = C$」
型 $\langle A, B, C\rangle = \langle int, int, int\rangle$
事例
$\langle 1,1,2\rangle, \langle 1,2,3\rangle, \langle 2,1,3\rangle, \langle 1,3,4\rangle, \langle 2,2,4\rangle,$
$\langle 3,1,4\rangle, \langle 1,4,5\rangle, \langle 2,3,5\rangle, \langle 3,2,5\rangle, \langle 4,1,5\rangle$

$minus(A, B, C)$：意味「$A - B = C$」
型 $\langle A, B, C\rangle = \langle int, int, int\rangle$
事例
$\langle 2,1,1\rangle, \langle 3,1,2\rangle, \langle 3,2,1\rangle, \langle 4,1,3\rangle, \langle 4,2,2\rangle,$
$\langle 4,3,1\rangle, \langle 5,1,4\rangle, \langle 5,2,3\rangle, \langle 5,3,2\rangle, \langle 5,4,1\rangle$

$order(A, B)$：意味「$A < B$」
型 $\langle A, B\rangle = \langle int, int\rangle$
事例
$\langle 1,2\rangle, \langle 1,3\rangle, \langle 1,4\rangle, \langle 1,5\rangle, \langle 2,3\rangle,$
$\langle 2,4\rangle, \langle 2,5\rangle, \langle 3,4\rangle, \langle 3,5\rangle, \langle 4,5\rangle,$

$equiv\_int(A, B)$：意味「$A = B$」
型 $\langle A, B\rangle = \langle int, int\rangle$
事例
$\langle 1,1\rangle, \langle 2,2\rangle, \langle 3,3\rangle, \langle 4,4\rangle, \langle 5,5\rangle$

$not\_equiv\_int(A, B)$：意味「$A \neq B$」
型 $\langle A, B\rangle = \langle int, int\rangle$
事例
$\langle 1,2\rangle, \langle 1,3\rangle, \langle 1,4\rangle, \langle 1,5\rangle, \langle 2,1\rangle, \langle 2,3\rangle, \langle 2,4\rangle,$
$\langle 2,5\rangle, \langle 3,1\rangle, \langle 3,2\rangle, \langle 3,4\rangle, \langle 3,5\rangle, \langle 4,1\rangle, \langle 4,2\rangle,$
$\langle 4,3\rangle, \langle 4,5\rangle, \langle 5,1\rangle, \langle 5,2\rangle, \langle 5,3\rangle, \langle 5,4\rangle,$

$equiv\_char(A, B)$：意味「$A = B$」
型 $\langle A, B\rangle = \langle char, char\rangle$
事例 $\langle a,a\rangle, \langle b,b\rangle, \langle c,c\rangle, \langle d,d\rangle, \langle e,e\rangle$

これらのデータを使ってメタ的性質に含まれるリテラルの最大個数が 2 個および 3 個について実験を行った．生成されたメタ的性質の一部を表 4 に示す．適切にメタ的性質が生成されていることが分かる．

自動生成に要した実行時間および生成された節の数を表 6. に示す．実行時間は十分実用的時間であった．ただしリテラル数を 2 から 3 にすると，実行時間，性質の個数ともに大きく増加している．探索の深さは性質の個数を爆発的に増加させていることが分かる．

表 4 に示された生成された性質の内，1〜13 および 16〜19，26，27 はあらかじめ予想していた節である．しかし，14 は頭部

表 3　実行時間と生成された節の数

最大リテラル数 [個]	2	3
実行時間 [sec]	0.266	38.656
節の数 [個]	208	23860
平均生成時間 [msec/個]	1.28	1.62

表 4　生成されたメタ的性質（抜粋）

（1） $order(A,B) \leftarrow add(A,\_,B)$
（2） $order(A,B) \leftarrow add(\_,A,B)$
（3） $order(B,A) \leftarrow minus(A,B,\_)$
（4） $order(B,A) \leftarrow minus(A,\_,B)$
（5） $minus(C,A,B) \leftarrow add(A,B,C)$
（6） $minus(C,B,A) \leftarrow add(A,B,C)$
（7） $minus(A,C,B) \leftarrow minus(A,B,C)$
（8） $add(B,A,C) \leftarrow add(A,B,C)$
（9） $add(B,C,A) \leftarrow minus(A,B,C)$
（10） $add(C,B,A) \leftarrow minus(A,B,C)$
（11） $equiv\_cha(A,A) \leftarrow equiv\_cha(A,\_)$
（12） $equiv\_cha(B,A) \leftarrow equiv\_cha(A,B)$
（13） $equiv\_cha(A,A) \leftarrow equiv\_cha(\_,A)$
（14） $equiv\_int(A,A) \leftarrow equiv\_int(A,\_)$
（15） $equiv\_int(B,A) \leftarrow equiv\_int(A,B)$
（16） $not\_equiv\_int(B,A) \leftarrow minus(A,\_,B)$
（17） $not\_equiv\_int(A,B) \leftarrow order(A,B)$
（18） $not\_equiv\_int(B,A) \leftarrow order(A,B)$
（19） $not\_equiv\_int(B,A) \leftarrow not\_equiv\_int(A,B)$
（20） $\square \leftarrow add(A,\_,B) \wedge add(A,B,A)$
（21） $\square \leftarrow add(A,C,B) \wedge add(A,B,C)$
（22） $\square \leftarrow add(A,B,C) \wedge minus(A,B,C)$
（23） $\square \leftarrow add(A,\_,\_) \wedge order(A,A)$
（24） $\square \leftarrow add(B,\_,A) \wedge order(A,B)$
（25） $\square \leftarrow add(\_,B,A) \wedge order(A,B)$
（26） $\square \leftarrow order(B,A) \wedge order(A,B)$
（27） $\square \leftarrow equiv\_int(A,B) \wedge not\_equiv\_int(A,B)$
（28） $\square \leftarrow not\_equiv\_int(A,\_) \wedge not\_equiv\_int(A,A)$

が同じ節が各 int 型の引数に対して作られた．これは *equiv_int* が反射律が成り立つためだと考えられる．これは，節の前件に無関係になるたつものであり，メタ的性質として冗長である．また，頭部が □ の節では，20，23，28 のように片方のリテラルのみで □ であることが判明する節が多く出力されていており，やはり冗長である．また，21，22 のように他の節を利用して導出可能な節も数多く出力された．ここに示したものは実験したどちらの場合にも出たメタ的性質である．これらの冗長な性質の導出がメタ的性質の爆発的増加の 1 つの原因であり，これらの制御は今後の検討課題である．

7. まとめと今後の課題

本研究では，述語の持つ意味的性質によって仮説の冗長性や充足不能性を判定できることに注目し，これを利用した仮説生成制御法について検討した．述語の持つ意味的性質をメタ的性質として定式化し，さらにメタ的性質の自動生成法を提案した．

本研究で提案したメタ的性質の自動生成は実用的観点から改善の余地がある．第一に，性質を大量に導出し，その中には，互いに導出可能な冗長な性質がある．また，性質として正しいとしても，これを利用する際に互いに呼び出し合い，無限ループに陥る性質がある．これらを考慮した自動生成法および仮説生成制御法を学習システムに組み込むことは今後の課題である．

文　献

[1] 古川康一，尾崎知伸，植野研：" 帰納論理プログラミング "，共立出版，2001
[2] J.R.Quinlan, R.M.Cameron-Jones :" Introduction of Logic Programs:FOIL and Related Systems ", New Generation Computing, vol 13, pp.287-312, OHMSHA, 1995
[3] J.R.Quinlan :"Learning Logical Definitions from Relations", Machine Learning, 5, pp.239-266, 1990
[4] L. Dehaspe, L. De Raedt:" Mining association rules with multiple relations ", ILP-97, LNAI1297,pp.125-132, 1997.
[5] L. Dehaspe, H. Toivonen :" Discovery of Relational Association Rules ", in Relational Data Mining, pp. 189–212, Springer, 2001.
[6] S. Muggleton:" Inverse Entailment and Progol ", New Generation Computing Journal, Vol. 13, pp. 245-286, 1995.
[7] 石田 浩之，中野 智文，犬塚 信博：" 述語のメタ的性質を利用した帰納学習の仮説制御 "，第二回 情報科学ワークショップ，2006