



HOKKAIDO UNIVERSITY

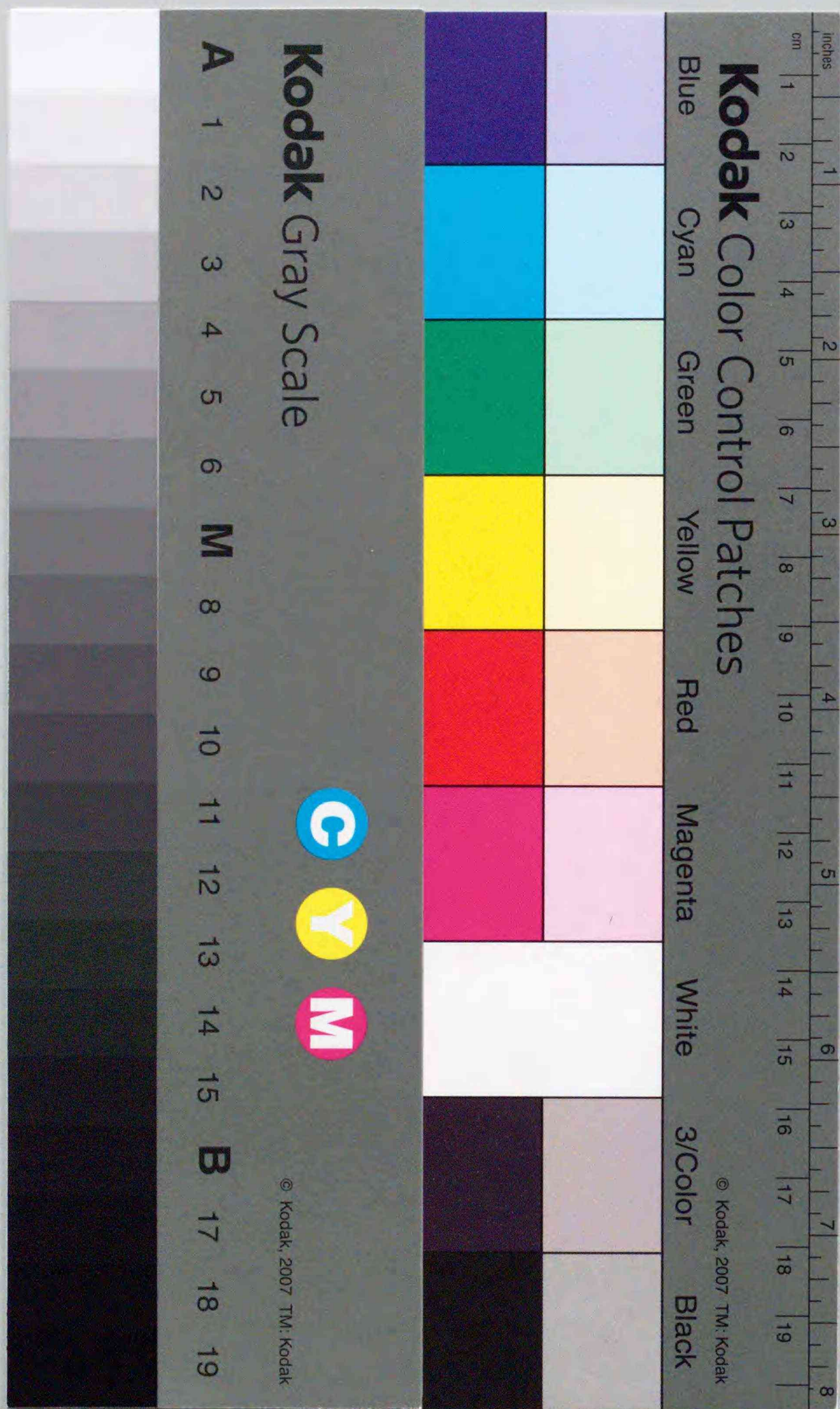
Title	帰納論理プログラミングにおける計算論的推論手法に関する研究
Author(s)	伊藤, 公人
Degree Grantor	北海道大学
Degree Name	博士(工学)
Dissertation Number	甲第4908号
Issue Date	1999-06-30
DOI	https://doi.org/10.11501/3154931
Doc URL	https://hdl.handle.net/2115/51619
Type	doctoral thesis
File Information	000000340119.pdf



帰納論理プログラミングにおける
計算論的推論手法に関する研究

平成11年 6月

伊藤 公人



帰納論理プログラミングにおける
計算論的推論手法に関する研究

平成11年 6月

伊藤 公人

目次

要旨	1
1 序論	3
1.1 計算論的学習理論	4
1.2 帰納推論	5
1.3 論理プログラミング	6
1.4 帰納論理プログラミング	7
1.5 本論文で提案する手法	9
1.5.1 論理的に正しい仮説の構成手法	9
1.5.2 質問を用いた論理プログラミング-言語学習における結果の応用	11
1.6 本論文の構成	12
2 論理プログラミング	15
2.1 論理プログラム	15
2.2 確定節プログラムの宣言的意味	18
2.3 確定節プログラムの手続き的意味	24
2.4 含意の証明	26
3 帰納論理プログラミング	29
3.1 帰納推論の基礎的概念	29
3.1.1 推論の対象になる規則の集合	30
3.1.2 規則の表現方法	30
3.1.3 推論を正しいと見る基準	32
3.1.4 推論の方法	33
3.2 帰納論理プログラミング	35
3.3 補助述語	36
3.4 背景知識を用いた構成的な仮説の生成問題	38
3.5 質問を用いた補助述語発見問題	39
4 論理プログラムを用いたアブダクションの実現	43
4.1 論理プログラムを用いた基礎事実のアブダクション	43
4.2 枚挙とSLD反駁を用いた自明なアブダクションアルゴリズム	44

4.3	SOLDR 導出を用いた構成的なアブダクション手法	45
4.4	失敗によるスキップ	47
4.5	SOLDR 導出の停止性問題とその解	48
4.6	SOLDR 導出の実装	48
4	章のまとめ	50
5	複数の例に共通する規則の発見手法	51
5.1	本章で扱う問題	51
5.2	底節交差汎化法の概要	52
5.3	仮説空間の構造化	54
5.3.1	包摂に基づく汎化	54
5.3.2	相対包摂に基づく汎化	54
5.4	底集合と底節	55
5.5	底節の生成手法	56
5.6	包摂に基づく共通汎化	56
5.7	底節交差汎化法とその性質	59
5.8	底節交差汎化法の利点	60
5.9	制限したクラスに対する底節交差汎化法	62
5.10	ILP システム・BORDA	64
5	章のまとめ	65
6	帰納論理プログラミングとしての正則言語の学習	67
6.1	正則言語とその学習に関する既知の定理	67
6.2	正則言語の学習アルゴリズム	69
6.3	正則言語を表現する論理プログラム	77
6.4	正則言語を表す論理プログラムの学習問題	78
6	章のまとめ	80
7	スキーマと質問を用いた述語発見による論理プログラムの構成的学習アルゴリズム	81
7.1	スキーマを用いた帰納論理プログラミング	81
7.2	学習アルゴリズムの戦略	82
7.3	準備	85
7.4	正則なルールスキーマ集合を持つ論理プログラム	88
7.5	学習アルゴリズム	89
7.5.1	サンプルタプルとテンポラリモデル	91
7.5.2	述語名決定木	92
7.5.3	述語名決定木の更新	92
7.5.4	サンプルタプルの追加	96
7.5.5	本章の主定理	96
7.6	定理 7.1 の証明	97
7.7	定理 7.2 の証明	97

7.7.1	定理 7.2 に関する補題	98
7.8	定理 7.3 に関する補題	98
7	章のまとめ	100
8	結論	101
8.1	底節交差汎化法	102
8.2	質問とスキーマを用いた述語発見による論理プログラムの学習手法	102
	謝辞	105
	付録	113
	A ILP システム・BORDA の全ソースプログラム	113

要旨

近年の計算機とそれを取りまくネットワーク環境の高度な発達により、種々のデータが計算機上に蓄積されつつある。蓄積されたデータは複雑かつ大量であるため、データの特徴づける規則を人手で得ることは困難である。そこで、計算機にデータの収集・蓄積・問い合わせをさせるだけでなく、データから新しい知識を学習または発見させる必要性が生じている。

本研究では、帰納論理プログラミングという枠組みを用いて、計算機に宣言的な知識を学習ならびに発見させる問題を扱う。宣言的な知識とは文章として表現されるような知識のことである。すなわち、人間にとって理解可能かつ再利用可能な知識を計算機によって獲得することを目指す。帰納論理プログラミングとは、データの集合からデータの特徴づける論理的な規則を計算する枠組みである。この枠組みを用いると、文の集合からの文法の学習、計算結果の集合からのプログラムの自動合成、関係データベースのタプルの集合からの問い合わせ文の自動生成など種々の学習や発見を統一的に議論できる。

現在、帰納論理プログラミングの主流は、相対包摂に基づく共通汎化を求める手法である。ところが、理論的な整備の遅れから、これらの手法が扱う規則には大きな制限がある。そこで本論文では、理論的背景が明らかである底節交差汎化法と呼ばれる手法を提案することにより、この制限を取り払う。その結果、従来のシステムより広いクラスに対する帰納論理プログラミングへの道が切り開かれる。

また、帰納論理プログラミングの研究において、新しい述語を自動生成する問題は困難であることが知られている。従来からスキーマを与えるなどの手法が提案されてきたが、本論文では、たとえスキーマを与えたとしても所与の具体例からの学習は非効率的であることを証明する。そこで、具体例に関する質問とスキーマを併用する新たな手法を提案し、この手法によって効率的な学習が可能となるクラスを明らかにする。すなわち、新述語の自動生成問題に対しても肯定的な結果を示す。

これを要するに本論文では、新たな帰納論理プログラミング手法である底節交差汎化法

を提案し、より広いクラスに対する帰納論理プログラミングが可能となることを示し、また、正則な論理プログラムによって表される関係はスキーマと質問により効率良く学習可能であることを明らかにした。

第 1 章

序論

近年の計算機とそれを取り巻くネットワーク環境の高度な発達により、種々のデータが計算機上で扱えるようになった。実際に各種の科学データが計算機上に蓄積されており、現在、各専門分野では実験結果のデータベース化が進められ、インターネットを通じてこれらのデータを問い合わせることも可能となってきた。蓄積されたデータの集合は複雑かつ大量であるため、それらのデータを特徴付ける規則を人手で得ることは困難である。そこで、計算機にデータの収集・蓄積・問い合わせをさせるだけでなく、データの集合から新しい知識を学習または発見させる必要性が生じている。

機械学習研究は、計算機などの機械に人間が持っているような学習する能力を与えることを目標とし、人工知能の分野において古くから研究されている。機械による知識発見はデータベースの大規模化にともない、近年活発に研究されはじめた。こうした中、機械による学習や発見を理論的に意味づけること、さらにそれらが本質的にどれだけの計算資源(時間や領域など)を必要とするのかという根本的な問題を解明することが注目されつつある。

本研究は、宣言的な知識の機械による学習や発見において、この問題の解明に寄与することを目標とする。宣言的な知識とは文章として表現されるような知識のことである。宣言的な知識の学習や発見を取り扱うことにより、人間にとって理解可能かつ再利用可能である知識を学習または発見することを目指す。

帰納論理プログラミング (Inductive Logic Programming, ILP) [Muggleton 91] は宣言的な知識に対する機械学習や知識発見の研究分野において、近年活発に研究されているテーマの一つである。帰納論理プログラミングは、一階述語論理を用いて機械学習や知識発見を実現することを目標とし、基礎から応用にわたる広い領域で研究が行われている [古川 97, 山本 97, 溝口 97, 沼尾 97]。論理プログラミングは一階述語論理の定理証明系をプログラミ

ングに応用したものであり、理論的にも実用的にも大きな研究成果を挙げたプログラミングパラダイムである。計算機科学と人工知能において、一階述語論理は宣言的な知識表現言語として古くから研究されてきた。したがって、一階述語論理は宣言的な知識に対する学習・発見の原理を追求する格好の土台である。一階述語論理を機械学習や知識発見における表現言語として用いる枠組みが帰納論理プログラミングである。

帰納論理プログラミングの初期の研究は、1970年代初期に始まっているが、活発に研究されるようになったのは論理プログラミングが成熟した1990年代に入ってからである。現在ではさまざまな帰納論理プログラミングシステムが計算機上で稼働し [Muggleton 94, Quinlan 96]、実践では新薬を発見する [Muggleton et.98] など優れた成果があがっている。しかし、その研究は強い実装指向に基づいたものが多く、未解決な理論的問題が多く残されている。本論文では帰納論理プログラミングが抱える現在の課題として次の二つを扱う。

[課題1] 学習や発見の仕組み、すなわち、理論的背景が明らかなシステムを構成する。

[課題2] 既に計算論的学習理論の分野で研究されてきた学習方式との関係を明らかにする。

本論文では、これらのそれぞれの課題に対して新しい帰納推論手法を提案することで解決する。

以下では、計算論的学習理論、帰納推論、論理プログラム、帰納論理プログラミングなどの本研究の背景について述べ、本論文で取り扱う具体的な問題について述べる。

1.1 計算論的学習理論

機械に学習させることは、情報科学における究極の目標の一つであり、従来から様々なアプローチによって研究がなされてきた。たとえば、記号処理系においては言語の学習や関数の学習などが古くから研究されてきた。また、例題や助言からの学習方式が提案され、それらの方式に基づくエキスパートシステムと呼ばれるシステムが構築された。一方、制御理論においてもニューラルネットなどの神経回路網モデルを用いた学習機能が研究されてきた。さらに、近年エージェントを利用した学習なども活発に研究されている。1980年代には、理論計算機科学においても学習に関するいくつかの主要な概念が提案され、様々な学習のためのアルゴリズムと計算量に関する研究が活発に行われた。

計算論的学習理論 (computational learning theory) [有川&西野 88] とは、これら様々な学習方式の根底に共通に存在するであろう「学習という行為の理論的原理」を解明し、「計算

1.2 帰納推論

機がある概念を学習したり新しい知識を発見したりするのに、本質的にどれだけの計算資源(時間や領域など)を必要とするのか」という根本的な問題を理論的に解明することを目指すとした研究分野である。計算論的学習理論は学習や発見の理論的原理を応用した「知的な計算機」を作るための理論的な準備として位置づけられる。

具体的には、計算論的学習理論では、形式言語理論、計算理論および一階述語論理など計算機科学における形式的手法を用いることにより、

1. 新しい学習モデルの導入
2. 学習に要する計算量の評価
3. 学習可能性・不可能性の証明

などを明らかにすることが研究目標となる。

1.2 帰納推論

計算論的学習理論が取り扱う学習モデルの中で、その基本となり歴史的にも古くから研究され発展してきたものに帰納推論 (Inductive Inference) がある。帰納推論ということは、論理学や哲学においても用いられてきたが、計算機科学においては「与えられたデータからそれを説明する規則を導き出す推論の総称」として使われている。

計算機科学における帰納推論研究の出発点は、1950年代の Moore による順序機械を入出力対から同定する研究 [Moore 56] に始まっていると言われている。その後、帰納的関数の帰納推論、言語の正データまたは正負データからの帰納推論 [Gold 67, Gold 78] などの初期の基礎的研究が行われた。言語の正負データからの帰納推論は、ちょうど人間の子供が母国語を文法を意識すること無く親などまわりの人間との会話を通して習得していく現象を形式化したものである。

形式言語を対象にした帰納推論については、Gold [Gold 78] や Angluin [Angluin 87] らにより学習可能性・不可能性が明らかにされ、質問を用いた言語学習など新しい計算モデルなどが導入された。形式言語はコンパイラの開発などの実用的な応用を含んでいたため、計算機科学では広く理論的に研究されており、それらの成果が形式言語の計算論的学習理論の様々な研究成果に結び付いている。

帰納推論の手法は大きくわけて、枚举的手法と構成的手法の2通りある。枚举的手法とは、仮説にある順序をつけて、その順序にしたがって仮説を枚举し、例を正しく説明する仮

説が見つかるまでそれを繰り返すという手法である。この手法は生成テスト法とも呼ばれ、現在の計算機の最も得意とする手法の一つである。一方、構成的手法とは、対象とする領域で成り立つ既に知られた規則を用いて、例を説明する仮説を導き出す手法である。これまで、帰納推論の研究の主流は枚挙的手法であるが、より知的な帰納推論を実現するためには構成的な帰納推論の手法が望まれる。

1.3 論理プログラミング

一階述語論理は、計算機科学と人工知能において形式言語とならんでよく用いられる知識の表現手法である。一階述語論理による知識表現は柔軟であり、仕様記述言語として古くから使われている。計算機上での定理の自動証明も古くから研究され続けており、大変成熟した研究領域である。特に、1930年のHerbrand[Herbrand 30]の結果に基づき、Robinsonが1965年に開発した融合規則(resolution rule)[Robinson 65]は、計算機上での自動証明の基本となっている。

論理プログラミングに関する研究は、定理自動証明の初期の成果を受ける形で1970年代に始まっている。Kowalski[Kowalski 74]とColmerauer[Colmerauer et. 73]は1972年に「論理をプログラミング言語として使うことができる」という基本的な考え方を提案した。この考え方が論理プログラミング(Logic Programming)である。つまり、論理には手続き的な解釈の方法があって、それにより論理が大変効果的なプログラミング言語となる[Kowalski 74]。それに先立ち1972年最初のPrologインタープリタが実装された[Roussel 75]。PrologはPROgramming in LOGicの頭文字をとったものである。現在では、Dec-10 Prolog, Quintus-Prolog, Sicstus-Prologなど優れたProlog処理系が登場し、その後並列処理などの進んだ技術も採り入れられている。

Kowalski[Kowalski 79a, Kowalski 79b]は論理プログラミングの基本的な考えが「アルゴリズムは論理と制御の二つの異なる部分からなる」ことを明らかにした。論理とは解くべき問題は何(what)であるかについての記述である。一方、制御とはそれをどう(how)解くかについての記述である。論理プログラミングが理想とするのは、プログラマはアルゴリズムの論理の部分のみ指定しさえすればよく、制御は論理プログラミングのシステムが勝手に実行してくれるような状況である。

論理にはプログラミング言語以外に、三つの解釈がある。一つは、データベースとしての解釈である[Lloyd 83, Ullman 88]。つまり、論理で書かれたプログラムはデータベース

と見なすことができ、その結果、関係データベースをとっても自然にかつ強力なものへと一般化できる。関係データベースは、基礎単位節(ground unit clause)のみから構成される論理プログラムに対応する。このように、データ、プログラム、問い合わせ、ビュー、保全性の制約を統一的に扱う言語として論理を使うという発想には、1.4節で述べるように大きな潜在的な可能性がある。

論理のもう一つの解釈として、プロセスとしての解釈がある。この解釈では、ゴールを並列に動くプロセスの系として考える。計算の1ステップは、一つのプロセスをプロセスの系(プロセスの呼び出しに適合した節の本体に現れるプロセス)へ変形することである。共有される変数は、プロセス間の通信チャンネルとして働く。このような解釈により、論理は、オペレーティング・システムへ応用することなどが可能であり、実際に日本の第5世代コンピュータ・プロジェクトでは、Prologを核にした新しいコンピュータアーキテクチャの開発が行われた。また、論理をシンセティック・メディア[長崎&田中 94]における部品の表現言語として用いる研究なども行われている[相笠 99, Hirata&Haraguchi et. 99]。このとき、部品間で共有される変数は、部品群で行われるデータの受け渡しを意味し、述語はその部品で計算されるべき結果を表している。

論理の三つ目の解釈として、形式言語としての解釈がある。論理プログラムを用いることにより、形式言語の生成規則を自然な形で表現することが可能であることが示されている[Arikawa et. 92]。

以上のような応用範囲の広さは、計算機科学において外見上は種々に異って見える側面に対し、論理プログラムが統一的枠組みになることを示している。

1.4 帰納論理プログラミング

帰納論理プログラミング(Inductive Logic Programming, 以下ILPと略す)は論理を用いて帰納推論を研究し実践する枠組みである。本節では、論理を用いて帰納推論を議論することの意義を述べる。

1.1節で述べた「学習や発見の原理」を解明するためには様々な学習方式を統一的に議論できる枠組みが必要であるが、1.3節で述べた理由から、計算機科学の道具において論理プログラムがその条件を満たす言語の一つであると考えられる。たとえば、大量かつ多様な情報が蓄えられたデータベースから、データ間で一般的に成り立つ規則を発見する手法の研究は、発見科学という新しい研究分野において広く関心が寄せられている。前節で述べ

たように論理プログラムはデータベースとみなすことができるので、データベースからの知識の発見も ILP を用いて扱うことができる。また、論理プログラムを用いて正則言語や文脈自由文法などの形式言語を表現することが可能であるので、古くから研究されている形式言語の帰納推論問題も ILP を用いて扱うことができる。

論理プログラミングにおいては、ユーザはアルゴリズムの論理の部分のみ指定しさえすればよく、制御は論理プログラミングのシステムが勝手に実行してくれる。したがって、論理プログラムを表現言語とする学習機械では、推論の対象となるアルゴリズムの論理の部分のみを推論すればよい。そしてそれは与えられた例に内在する規則の部分のみを推論するということであり、学習や発見の原理を追求するのに好都合である。また、帰納推論を他の言語などを用いて応用することを考えたときにも有用である。

ILP の研究は 1970 年代初頭の Plotkin による先駆的な研究のあとしばらく沈黙していたが、1982 年の Shapiro によるモデル推論システム [Shapiro 81] を皮切りに、論理プログラミングが成熟した 1980 年代後半に入り活発に研究が進み [Muggleton 91]、現在に至っている。ILP の研究は特に実装志向が強く、理論的にはまだ不備が残る。しかし、先に述べたように各種の機械学習を論理によって表現することができるので、本来の計算論的学習理論の目的にかなっており、その理論的整備が現在重要な課題となっている。

ILP を形式的に定義すると、与えられた正の具体例 (positive example) $E_1^{\oplus}, \dots, E_n^{\oplus}$ と負の具体例 (negative example) $E_1^{\ominus}, \dots, E_m^{\ominus}$ に対して

$$\begin{aligned} P \vdash E_1^{\oplus} \wedge \dots \wedge E_n^{\oplus} \\ P \not\vdash E_1^{\ominus} \vee \dots \vee E_m^{\ominus} \end{aligned} \quad (1.1)$$

となる論理プログラム P を計算することになる。あらかじめ学習者に論理プログラムで表された背景知識 (background knowledge) B が与えられている場合には

$$\begin{aligned} B \wedge H \vdash E_1^{\oplus} \wedge \dots \wedge E_n^{\oplus} \\ B \wedge H \not\vdash E_1^{\ominus} \vee \dots \vee E_m^{\ominus} \end{aligned} \quad (1.2)$$

となる節 H を計算するというものである。ここで、 $\vdash$ は一階述語論理における証明可能性を表す。

背景知識はプログラムで利用できるあらかじめ用意されたライブラリーと考えられるので、帰納推論に背景知識を使用することは自然である。また、人間がある新しい規則を発見する場合においても、ある種の背景知識を用いていると考えるのも自然なことである。人間の場合、背景知識はその人が以前に経験的に得たものである場合もあれば、先人たちが

導いて文書などに記録したものである場合もある。背景知識を用いない推論は人間でたとえると前者であり、背景知識を用いる場合は後者であると考えることができる。

帰納論理プログラミングの理論を計算論的学習理論として十分議論できるようなレベルに移行させるためには、その理論的解析、計算量解析、他の学習パラダイムとの関係を明白にする必要がある。

1.5 本論文で提案する手法

本論文では、帰納論理プログラミングにおける上述の二つの課題に対する解となる手法を構成する際の、各手法がみたすべき具体的な仕様を次のように設定する。まず、課題 1 に対しては「ある二つの具体例が与えられたときそれらの具体例とともに説明する一つの仮説を論理的に正しく構成する手法」を提案する。また、課題 2 に対しては「ある概念をあらわす規則を、質問をすることによって求める手法」を提案し、その計算論的解析を与える。

以下 1.5.1 節と 1.5.2 節とでそれぞれの問題について説明する。

1.5.1 論理的に正しい仮説の構成手法

人間は次のように観測事実を一般化することによって帰納推論を行って新しい知識を得る場合があると考えることができる。たとえば、

鉄片 A は 419 度まで熱すると溶けた
鉄片 B は 419 度まで熱すると溶けた
すべての鉄は 419 度まで熱すると溶ける

このような一般化は包摂に基づく共通汎化と呼ばれ、Plotkin により、複数の例から包摂に基づく共通汎化を計算するアルゴリズムが与えられている [Plotkin 70]。この仮説の生成手続きは構成的手法ということができる。なぜなら、「鉄片 A 」と「鉄片 B 」を「すべての鉄」に置き換えて、文末の動詞「溶けた」を「溶ける」に置き変えるだけで仮説を見つけることができるからである。もし、この場合に枚挙的な手法を用いるとすると、知っているすべての語からある文を作成し、それが与えられた例を説明しているかどうかをチェックすることになる。

さらに、次のような場合の一般化を考えてみる。

$$\frac{\begin{array}{l} \text{砂鉄を 419 度まで熱すると溶けた} \\ \text{空き缶を 419 度まで熱すると溶けた} \end{array}}{H}$$

ここで、次のような知識を既に持っているとする。以降、学習者があらかじめ持っている知識のことを背景知識と呼ぶ。

$$\begin{array}{l} \text{砂鉄は鉄である} \\ \text{空き缶は鉄でできている} \end{array}$$

すると、この背景知識を用いることにより、

$$\frac{\begin{array}{l} \text{砂鉄を 419 度まで熱すると溶けた} \\ \text{空き缶を 419 度まで熱すると溶けた} \end{array}}{\text{すべての鉄は 419 度まで熱すると溶ける}}$$

という一般化が可能となり仮説 H を得る。この種の一般化も構成的な手法であり、一般包摂に基づく共通汎化として、Buntine により提案された [Buntine 88]。

ここで、Buntine の一般包摂に基づく共通汎化の特徴を与えておく。この一般化では「溶ける」という概念の二つの例を背景知識を用いて一般化するものである。つまり、

$$\frac{\begin{array}{l} \text{溶ける} \leftarrow \text{鉄である} \leftarrow \text{砂鉄} \\ \text{溶ける} \leftarrow \text{鉄である} \leftarrow \text{空き缶} \end{array}}{\text{溶ける} \leftarrow \text{鉄である}}$$

という図式を用いて一般化を行っている。

しかし、新しい知識を発見する際、互いに異なる概念の例を用いることもある。たとえば、

$$\begin{array}{l} \text{砂鉄は鉄である} \\ \text{空き缶は鉄でできている} \\ \text{溶けると形が変わる} \\ \text{溶けると液体になる} \\ \text{液体は流れる} \end{array}$$

を背景知識として持っているときに

$$\frac{\begin{array}{l} \text{砂鉄を 419 度まで熱すると穴の空いた容器から流れ出た} \\ \text{空き缶を 419 度まで熱すると形が変わった} \end{array}}{\text{すべての鉄は 419 度まで熱すると溶ける}}$$

を推論する場合である。このような一般化は Plotkin によって導入され、**相対包摂に基づく共通汎化**と呼ばれている [Plotkin 71] が、相対包摂に基づく共通汎化を求める決定的な手法はこれまでに提案されていない。

本論文では、背景知識に関して、各具体例の相対包摂に基づく共通汎化を求める問題を扱う。

上の例に底節交差汎化法を適用すると

$$\frac{\begin{array}{l} \text{流れる} \leftarrow \text{溶ける} \leftarrow \text{鉄である} \leftarrow \text{砂鉄} \\ \text{形が変わる} \leftarrow \text{溶ける} \leftarrow \text{鉄である} \leftarrow \text{空き缶} \end{array}}{\text{溶ける} \leftarrow \text{鉄である}}$$

という図式を用いて一般化を行う。この場合には仮説はただ一つに定まるが、一般には仮説は複数個存在する。底節交差汎化法はそのような仮説をすべて求めることができる手法である。

底節交差汎化法で求まる仮説はすべて、与えられた正の例のいずれをも説明する。つまり、与えられた正の具体例 $E_1^{\oplus}, \dots, E_n^{\oplus}$ と背景知識 B に対して

$$B \wedge H \vdash E_1^{\oplus} \wedge \dots \wedge E_n^{\oplus}$$

となる仮説 H を構成的に帰納推論する。

現在、帰納論理プログラミングの主流は、相対包摂に基づく共通汎化を求める手法である。ところが、理論的な整備の遅れから、これらの手法が扱う規則には大きな制限がある。そこで本論文では、底節交差汎化法を提案することにより、この制限を取り払う。また、底節交差汎化法に関連する研究として RLGG [Buntine 88] や CIG_{≥Σ} [Page&Frish 92] がある。両者とも述語記号が異なるような具体例を扱うことができない。それに対して、底節交差汎化法では述語記号が異なるような具体例を扱うことができる。

1.5.2 質問を用いた論理プログラミング-言語学習における結果の応用

先に、背景知識を用いない ILP とは与えられた正の具体例 $E_1^{\oplus}, \dots, E_n^{\oplus}$ と負の具体例 $E_1^{\ominus}, \dots, E_m^{\ominus}$ に対して、条件 1.1 を満たす論理プログラム P を計算することであると述べた。しかし、Gold と Angluin の形式言語の帰納推論の研究結果は、正則言語は与えられた正負の例からではその言語を表現する有限オートマトンを効率よく帰納推論できないこと [Gold 78]、質問を用いることにより効率的な帰納推論が可能となること [Angluin 87] を示し

ている。正則言語は論理プログラムで簡単に記述できる。したがって、論理プログラムの帰納推論においても、与えられた正負の例からでは効率よく帰納推論できないが、質問を用いることにより効率的な帰納推論が可能となるクラスが存在する。Sakakibara[Sakakibara 90] は木オートマトンを用いて正則言語の質問学習を一般化している。本論文では正則言語や木オートマトンの学習問題を帰納論理プログラミングの観点から見たときに、これらの学習アルゴリズムがどのような論理的性質を用いているのかを明らかにし、帰納論理プログラミングのための一般的な手法を導く。

Angluin の学習アルゴリズムを論理プログラムの学習の観点でみれば、学習者が補助述語を構成するとみなすことができる。補助述語 (auxiliary predicate) とは学習の対象となる関係 R を表す述語以外の述語のことである。関係 R を論理プログラムで表現するとき、補助述語なしのプログラムでは表現できないが、補助述語を用いたプログラムでは表現できる場合が知られている [Stahl 95]。補助述語の定義を自動的に発見する問題は、膨大な探索空間を必要とする困難な問題であり、様々なアプローチから研究が行われている [石坂 89, Stahl 95, Flach 93, Mofizur&Numao 94]。例えば、Shapiro のモデル推論システム (MIS)[Shapiro 81] では、矛盾点追跡アルゴリズム中で補助述語に対する質問を用いてこの問題を回避している。Muggleton の Progol[Muggleton 94, Muggleton 95] では、補助述語に背景知識としてあらかじめ学習システムに与えられたものを使っている。

補助述語を効率よく発見するための一つのアプローチとして、あらかじめプログラムのスキーマ (schema) を与える手法が提案されており [Stahl 95, Flener 96]、本論文はこの手法に注目する。具体的には、CIA[Raedt&Bruynooghe 92] や RDT[Wirth&O'Roke 91] と同様にスキーマを二階の論理における述語変数を用いた節の形で表す。すると、先の正則言語の学習も木オートマトンの学習も Even-linear 言語の学習 [Takada 88] もスキーマを用いた述語発見による学習問題となる。本論文では、これら言語学習の結果を応用して、帰納論理プログラミングにおけるスキーマを用いた補助述語の発見問題に対する解を与える。

1.6 本論文の構成

本論文では、2 章で一階述語論理や論理プログラミングに関する準備を行う。3 章で、帰納推論や帰納論理プログラミングについて準備を行い、本論文で扱う問題を定義する。4 章で論理プログラムを用いたアブダクションの実現手法である SOLDRA 導出を提案し、この結果を用いて 5 章では、論理的に正しい仮説の構成手法である底節交差汎化法を提案し、そ

の理論的な議論を行う。さらに同手法に基づいて BORDA という帰納論理プログラミングシステムを計算機上に実装する。6 章では、正則言語の学習を帰納論理プログラミングの観点から考察する。7 章では、6 章の議論に基づいて、スキーマと質問を併用した帰納論理プログラミング手法を提案する。最後に 8 章で、本論文のまとめを述べる。

第 2 章

論理プログラミング

本章では一階述語論理や論理プログラミングに関する準備を行う。

論理プログラミングに関する研究は 1970 年代の初頭に始まったが、それは自動定理証明と人工知能の初期の仕事が直接的にもたらしたものであると言われている [Lloyd 84]。自動定理証明に関する重要な成果は、1965 年の融合規則 (resolution rule) を導入した Robinson による歴史的論文 [Robinson 65] である。融合規則を用いた導出は計算機上での自動化に適した推論規則である。さらに、Kowalski は融合規則による推論に手続き的な解釈 (procedural interpretation) を持たせることによって、論理がプログラミング言語となることを示した [Kowalski 74]。

本章では、融合導出に基づいた論理プログラミングのシステムについて説明し、帰納論理プログラミングによる構造的な仮説発見のための準備を行う。

2.1 論理プログラム

本節では論理プログラムの構文について準備を行う。最初に一般的な一階述語論理の論理式 (well-formed formulas) の構文を導入し、次に論理プログラムとして確定節プログラムを定義する。

一階理論 (first order theory) で用いるにアルファベットを次のように準備する。

定義 アルファベット (alphabet) は記号の七つのクラスからなる。

1. 変数 (variable)
2. 定数 (constant)
3. 関数 (function)

4. 述語 (predicate)
5. 結合子 (connective)
6. 限定記号 (quantifier)
7. 句読点記号 (punctuation symbol)

本論文では、変数、定数、関数、述語について次のような略式的な表記上の約束を用いる。変数は通常 X, Y, Z, W の文字やこれらに添字をつけたもので表記する。定数は通常 a, b, c, d の文字やこれらに添字をつけたもので表記する。関数は通常 f, g, h の文字やこれらに添字をつけたもので表記する。また、述語は通常 p, q, r の文字やこれらに添字をつけたもので表記する。さらに、変数の集合を V と、関数記号の集合を F とそして述語記号の集合を Π と表記する。定数記号は0引数の関数記号とみなす。結合子には $\neg, \wedge, \vee, \leftarrow, \leftrightarrow$ などがあり、限定記号には $\exists, \forall$ がある。最後に句読点記号は、(と) と , である。

アルファベットにより与えられる一階言語を項、論理式を用いて定義する。

定義 項 (term) は次のように帰納的に定義される。

1. 変数は項である。
2. 定数は項である。
3. f が n 引数の関数、 $t_1, \dots, t_n$ が項であるならば、 $f(t_1, \dots, t_n)$ は項である。
4. 以上のことから項であるとわかるもののみが項である。

関数記号および定数記号の集合 F と変数の集合 V から作られる項の集合を $T(F \cup V)$ 、 F のみから作られる項の集合を $T(F)$ と表す。 $T(F)$ の要素を基礎項 (ground term) という。 $F|_n$ で F 中の n 引数の関数記号の集合を表す。定数記号の集合は $F|_0$ で表すものとする。 Π についても同様である。 $T(F)^n$ で n 個の $T(F)$ の直積を表し、 $T(F)^* = T(F)^1 \cup T(F)^2 \cup \dots$ とする。 $T(F)^n$ の部分集合を基礎項の n 項関係という。

本論文の例などで用いる $[A|X]$ などのリストを表す項の表現は Prolog の標準的な記法 [Clocksin&Mellish 94] に従うとする。定数 $[]$ は空リスト nil を表し、項 $[A|X]$ はリスト構成子 $'$ を用いた項 $.(A, X)$ を表す。たとえば、 $[a|[b|[c][]]]$ はリスト $.(a, .(b, .(c, nil)))$ を表す。しばしば、簡単のため $[a|[b|[c][]]]$ を $[a, b, c]$ のように略記する。

定義 論理式 (well-formed formulas) は、つぎのように帰納的に定義される。

1. $p \in \Pi|_n$ かつ $\langle t_1, \dots, t_n \rangle \in T(F \cup V)^n$ ならば、 $p(t_1, \dots, t_n)$ は論理式である。この型のものを原子論理式 (atomic formula)、または簡単に原子式 (atom) という。変数を含まない原子式は基礎原子式 (ground atom) と呼ばれる。
2. F と G が論理式ならば、 $(\neg F)$ 、 $(F \wedge G)$ 、 $(F \vee G)$ 、 $(F \leftarrow G)$ 、 $(F \leftrightarrow G)$ は、論理式である。
3. F が論理式で X が変数ならば、 $\forall X(F)$ 、 $\exists X(F)$ は論理式である。
4. 以上のことから論理式であるとわかるもののみが論理式である。

本論文では、便宜上 $T(F \cup V)^n$ の要素 $\langle t_1, \dots, t_n \rangle$ を $\mathbf{t}$ と表記することがある。このとき、述語記号 $p \in \Pi|_n$ に対して $p(\mathbf{t})$ は原子式 $p(t_1, \dots, t_n)$ を意味する。次に一階言語を定義する。
定義 アルファベットにより与えられる一階言語 (first order language) は、アルファベットの記号から構成されるすべての論理式の集合からなる。

例 2.1 $(\forall X(\exists Y(q(X) \leftarrow p(X, Y))))$ は論理式である。句読点記号の括弧をはずすと、この論理式は $\forall X \exists Y(q(X) \leftarrow p(X, Y))$ のと書くことができる。

限定記号と結合子の意味は次のようになる。 $\neg$ は否定 (negation)、 $\wedge$ は論理積 (conjunction/and)、 $\vee$ は論理和 (disjunction/or) である。 $\leftarrow$ は含意 (implication) であり、 $'F \leftarrow G'$ は「 G が成り立つならば F が成り立つ」ことを意味する。 $\leftrightarrow$ は同値 (equivalence) であり、 $'F \leftrightarrow G'$ は「 G が成り立つならばかつそのときに限り F が成り立つ」ことを意味する。また、 $\exists$ は存在記号 (existential quantifier) であり、 $'\exists X(F)'$ は「 F が成り立つ X のある値が存在する」を意味する。 $\forall$ は全称記号 (universal quantifier) であり、 $'\forall X(F)'$ は「 X のすべての値について F が成り立つ」を意味する。

定義 $\exists X$ の $\exists X(F)$ 中での有効範囲 (scope) は F である。 $\forall X$ の場合も同様である。論理式中の変数 X の出現 (occurrence) が束縛 (bound) されているとは、限定記号の直後に X が出現しているか、 $\exists X$ または $\forall X$ の有効範囲内に X が出現している場合のいずれかである。これ以外の変数の出現を自由 (free) であるという。すべての変数の出現が束縛されている論理式を閉論理式 (closed formula) という。

定義 確定節 (definite clause) とは次の形をした閉論理式である。

$$C = \forall X_1 \dots X_k (A_0 \vee \neg A_1 \vee \dots \vee \neg A_n).$$

ここで, $A_0, \dots, A_n$ は原子式であり, $X_1, \dots, X_k$ は原子式 $A_0, \dots, A_n$ 中に出現するすべての変数である.

便宜上, n が無限であるような C も論理式として扱い, C も確定節とよぶ.

定義 確定節の連言を**確定節プログラム** (definite program) という. 確定節プログラム $P = C_1 \wedge \dots \wedge C_n$ を, しばしば集合 $P = \{C_1, \dots, C_n\}$ で表す.

定義 ゴール節 (goal clause) とは次の形をした式である.

$$D = \forall X_1 \dots X_k (\neg A_1 \vee \dots \vee \neg A_n).$$

ここで, $A_0, \dots, A_n$ は原子式であり, $X_1, \dots, X_k$ は原子式 $A_0, \dots, A_n$ 中に出現するすべての変数である.

確定節またはゴール節を**ホーン節**という. ホーン節 $\forall X_1 \dots X_k (F)$ に対して $X_1 \dots X_k$ を省略し, 単に $\forall (F)$ と書く場合もある.

論理プログラミングにおいて確定節は頻繁に出てくるので, 確定節とゴール節を以下のような含意の形で書く.

$$C = A_0 \leftarrow A_1, \dots, A_n,$$

$$D = \leftarrow A_1, \dots, A_n.$$

この記法は, 論理式の変形によって得ることができるが, $A_1, \dots, A_n$ における句読点「;」は仮定の部分にある原子式の論理積を表現していることに注意されたい.

原子式 A_0 を C の**頭部**, 原子式の列 $A_1, \dots, A_n$ を C や D の**本体**という. 確定節 C は本体を持たないとき, **単位節** (unit clause) または**事実** (fact) と呼ばれる. また, C は本体を持つとき, **ルール節** (rule clause) または**ルール** (rule clause) と呼ばれる. 確定節 $C = A_0 \leftarrow A_1, \dots, A_n$ に対して, C^+ で単位節 $A_0 \leftarrow$ を表し, C^- でゴール節 $\leftarrow A_1, \dots, A_n$ を表すとする.

定義 本体を持たないゴール節を**空節** (empty clause) といい, $\square$ で表す. 空節は矛盾と考える.

2.2 確定節プログラムの宣言的意味

定義 一階言語 L に対して, 以下の4つを定めると**解釈** (interpretation) が1つ定まる.

1. 解釈の**領域** (domain) と呼ばれる空でない集合 D
2. L 中の各定数に対する D の要素の割り当て

3. L 中の各 n 引数関数に対する D^n から D への写像の割り当て

4. L 中の各 n 引数述語に対する D^n から $\{\text{真}, \text{偽}\}$ への写像 (すなわち D^n 上の関係) の割り当て

定義 I を一階言語 L の解釈とする. I に関する**変数割り当て** (variable assignment) とは, L 中の各変数へ I の領域の要素を割り当てることである.

定義 I を一階言語 L の領域 D をもつ解釈とし, A を変数割り当てとする. L 中の項への (I と A に関する) **項割り当て** (term assignment) とは, 次のように定義される.

1. 各変数には A にしたがって割り当てが行われる.
2. 各定数には I にしたがって割り当てが行われる.
3. $t'_1, \dots, t'_n$ を $t_1, \dots, t_n$ への項割り当てとし, f' を f へのわりあてとすると, $f'(t'_1, \dots, t'_n)$ は $f(t_1, \dots, t_n)$ への項割り当てである.

定義 I を一階言語 L の領域 D をもつ解釈とし, A を変数割り当てとする. すると論理式には, I と A に関する**真理値** (truth value), 真または, 偽を次のように与えることができる.

1. 論理式が原子式 $p(t_1, \dots, t_n)$ ならば, 真理値は $p'(t'_1, \dots, t'_n)$ を計算して得られる. ここで, p' は I により p へ割り当てられた写像で, $t'_1, \dots, t'_n$ は I と A による $t_1, \dots, t_n$ への項割り当てである.
2. 論理式が $\neg F$, $F \wedge G$, $F \vee G$, $F \leftarrow G$ ないし $F \leftrightarrow G$ の形をしているならば, その真理値は次の表に従う.

F	G	$\neg F$	$F \wedge G$	$F \vee G$	$F \leftarrow G$	$F \leftrightarrow G$
真	真	偽	真	真	真	真
真	偽	偽	偽	真	真	偽
偽	真	真	偽	真	偽	偽
偽	偽	真	偽	偽	真	真

3. 論理式が $\exists X (F)$ の形をしているならば, その真理値が真になるのは, ある $d \in D$ が存在して F の真理値が I と $A(X/d)$ に関して真になるときである ($A(X/d)$ は X に d が割り当てられる以外は A を同じ割り当てである). それ以外の場合は真理値は偽となる.

4. 論理式が $\forall X(F)$ の形をしているならば、その真理値が真になるのは、すべての $d \in D$ に対して F の真理値が I と $A(X/d)$ に関して真になるときである。それ以外の場合は真理値は偽となる。

明らかに閉論理式の真理値は変数割り当てには依存しない。したがって、ある解釈に関する閉論理式の真理値という言い方に曖昧なところはない。

定義 I を一階言語 L の解釈とし、 F を L の閉論理式とする。 I に関する F の真理値が真のとき、 I は F のモデル (model) であるという。 F のモデルとなる L のある解釈 I が存在するとき、 F は充足可能 (satisfiable) であるという。 L のすべての解釈 I について、 I が F のモデルであるとき、 F は恒真 (valid) であるという。 F にモデルが存在しないときは、 F は充足不可能 (unsatisfiable) であるという。

定義 F, G を一階言語 L の閉論理式とする。 L のすべての解釈 I について、

- I が F のモデルならば I は G のモデルになる

とき、 G は F の論理的帰結であるといい、 $F \models G$ と書く。

命題 2.1 F, G を一階言語 L の閉論理式とする。すると、つぎの2つの事柄は同値である。

1. $F \models G$ である。
2. $F \wedge (\neg F)$ が充足不可能である。

定義 L を、関数および定数の集合が F であり、述語の集合が Π である一階言語とする。基礎項の集合 $T(F)$ をエルブラン空間 (Herbrand universe) といい、 U_L と表記する。 $(L$ に定数が無い場合は、たとえば a のようななんらかの定数を付け加えて基礎項を作れるようする。) Π と F から作られる基礎原子式の集合をエルブランエルブラン基底 (Herbrand base) といい、 B_L と表記する。つまり、

$$B_L = \{p(t_1, \dots, t_n) \mid p \in \Pi, \langle p_1, \dots, p_n \rangle \in T(F)^n, n \text{ は自然数}\}$$

である。

定義 L を一階言語とする。 L の解釈 I がエルブラン解釈 (Herbrand interpretation) であるとは、次の条件を満たすときをいう。

1. 解釈 I の領域がエルブラン空間 U_L である。

2. L の定数に対して U_L 中の自分自身が割り当てられている。
3. f が L の n 引数関数ならば、 f には $(t_1, \dots, t_n) \rightarrow f(t_1, \dots, t_n)$ で定義される $(U_L)^n$ から U_L への写像が割り当てられている。

一階言語 L の閉論理式 F のモデルであるエルブラン解釈を F のエルブランモデル (Herbrand model) という。

エルブラン解釈の定義では、 L 中の述語への割り当て方については何も述べていない。そのため、その割り当ての仕方を変えることで、異ったエルブラン解釈が生じる。ところでエルブラン解釈に関しては定数と関数の割り当て方が決まっているので、任意のエルブラン解釈はあるエルブラン領域の部分集合と同一視することが可能である。すなわち、任意のエルブラン解釈に対応するエルブラン領域の部分集合とは、その解釈について真となるすべての基礎原子式の集合とする。逆にエルブラン領域の任意の部分集合を与えると、対応するエルブラン解釈は $p(t_1, \dots, t_n)$ が与えられた部分集合の中に入るときにのみ、 $p(t_1, \dots, t_n)$ は真にすると決めることで定義される。このようにエルブラン解釈とエルブラン領域の部分集合との同一視を以後行っていく。

定義 代入 (substitution) θ とは、 $\{v_1/t_1, \dots, v_n/t_n\}$ の形をした集合である。ここで、 $v_1, \dots, v_n$ は変数、各 t_i は v_i とは異なる項であり、 $v_1, \dots, v_n$ は互いに異なる変数である。各要素 v_i/t_i は変数 v_i の束縛 (binding) とよばれる。 $t_1, \dots, t_n$ がすべて基礎項であるとき、 θ を基礎代入 (ground substitution) という。

定義 代入 θ を $\{v_1/t_1, \dots, v_n/t_n\}$ 、 E を項、リテラル、または、リテラルの論理和または論理積とする。このとき、 $E\theta$ は E 中の変数 $v_1, \dots, v_n$ をそれぞれ $t_1, \dots, t_n$ におきかえて得られる項、リテラル、または、リテラルの論理和または論理積であり、 E の代入例 (instance) と呼ばれる。 $E\theta$ に変数が含まれないときは、 $E\theta$ を E の基礎代入例 (ground instance) であるという。

定義 代入 $\theta = \{u_1/s_1, \dots, u_m/s_m\}$ 、 $\gamma = \{v_1/t_1, \dots, v_n/t_n\}$ に対して、 θ と γ の合成とは代入の合成 $\theta\gamma$ とは集合

$$\{u_1/s_1\gamma, \dots, u_m/s_m\gamma, v_1/t_1, \dots, v_n/t_n\}$$

から束縛 $u_i/s_i\gamma$ で $u_i = s_i\gamma$ となるものと、束縛 v_j/t_j で $v_j \in \{u_1, \dots, u_m\}$ となるものを除いてできる代入のことである。

論理プログラムにおいて次のスコールム化代入という考え方はしばしば、有用である。

定義 ホーン節 C に中に現れる変数 $X_1, \dots, X_k$ に対して, L に新しい定数 c_{X_i} を導入して各変数 X_i を c_{X_i} に置き換える代入 σ をスコーム化代入という.

定義 また, 節 C が D の変種であるとは, ある代入 θ, σ が存在して $C\theta = D$ かつ $C = D\sigma$ となることをいう.

定義 $S = e_1, e_2, \dots, e_n$ を項または原子式の有限集合とする.

$$e_1\theta = e_2\theta = \dots = e_n\theta$$

となるような代入 θ を S の単一化子という. S の各単一化子 θ' に対して $\theta' = \theta\gamma$ となるような代入 γ が存在するとき, θ は S の最汎単一化子であるという.

最汎単一化子を求めるさまざまなアルゴリズムが提案されている. 代表的なアルゴリズムについては文献 [Lloyd 84] を参照されたい.

定義 P をプログラム, $G \leftarrow A_1, \dots, A_k$ をゴールとする. $P \wedge G$ の解代入 (answer substitution) とは G の変数への代入のことである. $\forall((A_1 \wedge \dots \wedge A_k)\theta)$ が P の論理的帰結であるとき, θ を $P \wedge G$ の正解代入 (correct answer substitution) という.

論理プログラムの正解代入は最小エルブランモデルとよばれるプログラムの記号を用いた意味で特徴づけられる.

定義 P のすべてのエルブランモデルの交わりを最小エルブランモデル (least Herbrand model) といい, $M(P)$ と表記する.

P をプログラムとする. すると,

$$M(P) = \{A \in B_P \mid A \text{ は } P \text{ の論理的帰結}\}$$

であることが知られている [Lloyd 84].

次に $M(P)$ の求め方を与える.

定義 P を確定節プログラムとする. P による被覆写像 $T_P: 2^{B_P} \rightarrow 2^{B_P}$ を次のように定義する.

I を基礎原子式の集合とすると,

$$T_P(I) = \{A_0 \in B_P \mid A_0 \leftarrow A_1, \dots, A_n \text{ は } P \text{ 中の節の基礎代入例で } \{A_1, \dots, A_n\} \subseteq I\}$$

次の定理で, P の最小エルブランモデル $M(P)$ を P による被覆写像 T_P で特徴付けることができる.

定理 2.1 [Lloyd 84] P を確定節プログラムとすると, P の最小エルブランモデル $M(P)$ は $T_P \uparrow \omega$ に等しい. ここで, ω は最初の無限順序数とする.

この定理の証明 [Lloyd 84] は順序数冪の概念を必要とするためその詳細について本論文では触れず, 非形式的だが直感的な説明をするにとどめる. $T_P \uparrow \omega$ の意味は次のとおりである.

$$T_P \uparrow 0 = \phi(\text{空集合})$$

$$T_P \uparrow n = T_P(T_P \uparrow n - 1)$$

$$T_P \uparrow \omega = \bigcup_{i=0}^{\infty} T_P(T_P \uparrow n - 1)$$

つまり, この定理は P の最小エルブランモデルは T_P の最小不動点に等しいことを示している. このようにして T_P を用いて P の論理的帰結となる基礎原子式を求める手法は, ボトムアップ評価 (bottom-up evaluation) と呼ばれている.

ボトムアップ評価の様子を次の例で示す.

例 2.2 次の確定節プログラム P を考える.

$$P = \left\{ \begin{array}{l} \text{odd}(0) \leftarrow \\ \text{odd}(s(s(X))) \leftarrow \text{odd}(X) \end{array} \right\}$$

P から $M(P)$ を作る手順は次の通りである.

$$T_P \uparrow 0 = \phi$$

$$T_P \uparrow 1 = \{\text{odd}(s(0))\}$$

$$T_P \uparrow 2 = \{\text{odd}(s(s(s(0)))), \text{odd}(s(0))\}$$

$$T_P \uparrow 3 = \{\text{odd}(s(s(s(s(s(0)))))), \text{odd}(s(s(s(0)))), \text{odd}(s(0))\}$$

$\vdots$

$$T_P \uparrow \omega = \{\text{odd}(s^n(0)) \mid n = 1, 3, 5, 7, \dots\}$$

定数 0 を自然数の 0, $s(0)$ を自然数 1, $s(s(0))$ を自然数 2 のように, 関数 s を次の自然数を表す後続関数とみなしたとき, 述語 odd は奇数全体を表す.

以下の定理は論理プログラミングにおいて良く知られている.

定理 2.2 [Lloyd 84] P を確定節プログラム, G をゴール $\leftarrow A_1, \dots, A_k$ とする. $P \wedge G$ の解代入 θ は, $(A_1 \wedge \dots \wedge A_k)\theta$ を基礎にすると仮定する. すると以下の事柄は同値である.

1. θ は正解代入
2. $(A_1 \wedge \dots \wedge A_k)\theta \in M(P)$

この定理の証明は論理プログラミングの解説書 [Lloyd 84] を参照されたい。

2.3 確定節プログラムの手続き的意味

本節では論理プログラムの手続き的意味論に関して準備を行う。つまり、あるゴール G の論理プログラム P に関する正解代入をえる手続き SLD 導出について述べる。SLD 導出は論理を Prolog などの計算機言語で扱うための基礎となる仕組みである。本節では SLD 導出の基本的定義と性質を与え、4章で議論する SOLDR 導出ための準備を行う。

導出推論規則に基づく反駁手続きは数多く存在し、その多くは Robinson による本来の手続きを改良したものである。ここで扱う SLD 導出もそのひとつである。SLD 導出とは、確定節 (Definite clause) の選択関数 (Selection function) 付きの線形導出 (Linear resolution) を示している。

定義 計算規則 (computation rule) とは、ゴール節の集合から原子式の集合への関数で、あるゴールに対するこの関数の値は、常にそのゴール中のある原子式となるものである。その原子式を選択された原子式 (selected atom) と呼ぶ。

定義 P を確定節プログラム、 G をゴール節、 R を計算規則とする。3つ組 (G_i, θ_i, C_i) の有限または無限列は、以下の条件を満たすとき $P \wedge G$ の **SLD 導出** (derivation) という。

1. G_i はゴール節、 θ_i は代入、 C_i は P 中の節の変種であり、 $i \neq j$ ならば C_i と C_j は変数を共有しない。
2. $G_0 = G$ 。
3. 任意の i について、 $G_i = \leftarrow A_1, \dots, A_m$ であり、 A_j が計算規則 R によって選択された原子論理式であるすると、 $G_{i+1} = \leftarrow (A_1, \dots, A_{j-1}, A'_1, \dots, A'_k, A_{j+1}, \dots, A_m)\theta_i$ である。ここで、 θ_i は C_i^+ と $A_j \leftarrow$ の最汎単一化子であり、 $C_i^- = \leftarrow A'_1, \dots, A'_k$ である。

定義 $P \wedge G$ の R による **SLD 反駁** (refutation) とは、最後のゴールとして空節 $\square$ をもつ、 R による $P \wedge G$ の有限な SLD 導出のことである。もし、 $G_n = \square$ ならば、反駁の長さが n であるという。

定義 P を確定節プログラム、 G をゴール、 R を計算規則とする。 $P \wedge G$ の R 計算解代入 (R -computed answer substitution) θ とは、 R による $P \wedge G$ の SLD 反駁で使われた最汎単一化子 $\theta_1, \dots, \theta_n$ の合成を G に現れる変数へ制限した代入のことである。

例 2.3 例 2.2 のプログラム P におけるゴール $\leftarrow \text{odd}(s^5(0))$ の SLD 反駁は成功する。その様子を図 2.1 に示す。

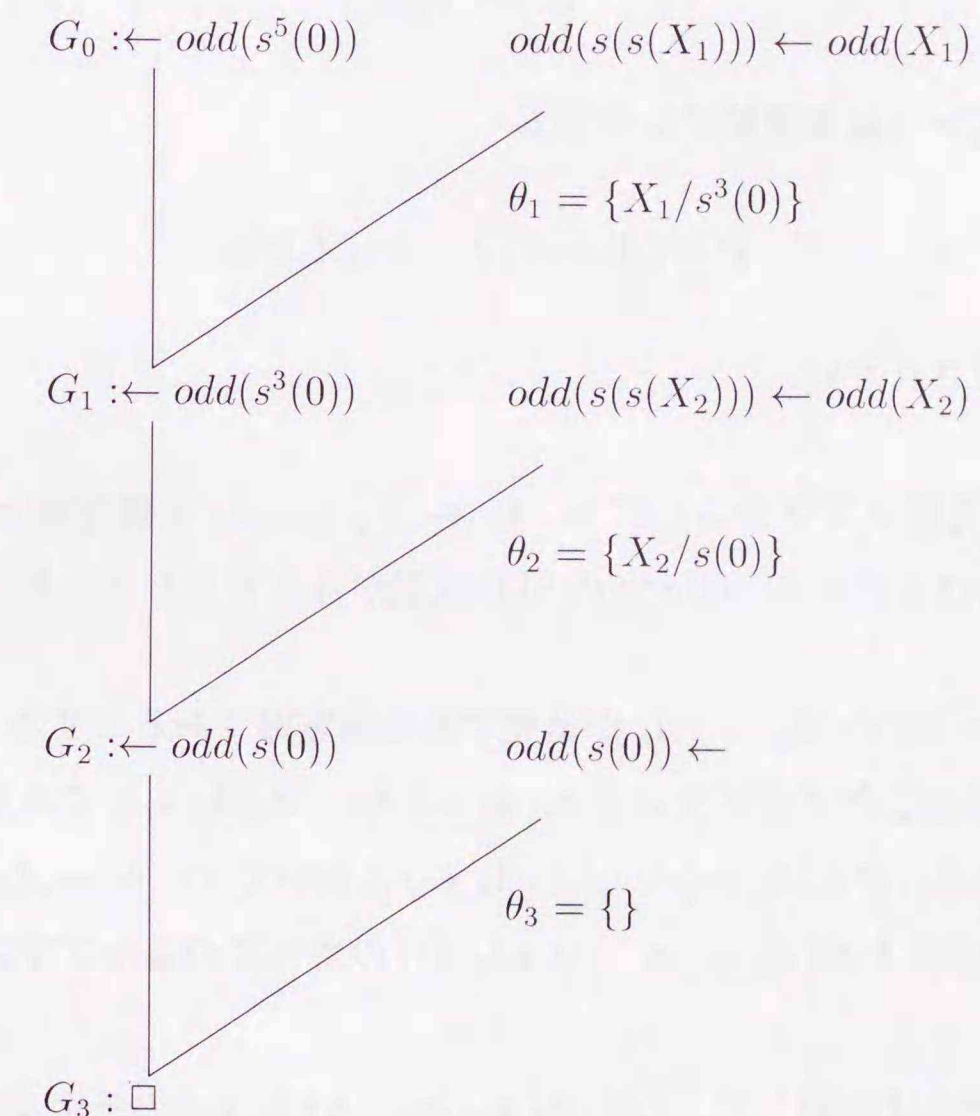


図 2.1: SLD 反駁

定理 2.3 [Lloyd 84] (SLD 導出の健全性)

P を確定節プログラム、 G をゴール、 R を計算規則とすると、 $P \wedge G$ のすべての R 計算解代入は、正解代入である。

定理 2.4 [Lloyd 84] (SLD 導出の完全性)

P を確定節プログラム、 G をゴールとする。 $P \wedge G$ のすべての正解代入 θ について、ある計算規則 R と、 $P \wedge G$ の R 計算解代入 θ' と $\theta = \theta'\gamma$ をみたす代入 γ が存在する。

2.4 含意の証明

これまでは一般的な論理プログラミングの概念を準備してきた。本節では第5章の準備のために、節 C がプログラム P の論理的帰結であるための条件を与える。

確定節 $C = A_0 \leftarrow A_1, \dots, A_n$ に対して、 σ をスコールム化代入とする。 $C\sigma$ には変数が含まれていないので、 $\neg(C\sigma^+)$ はゴール節

$$G = \leftarrow A_0\sigma$$

となること、 $\neg(C\sigma^-)$ は確定節プログラム

$$P = (A_1\sigma \leftarrow) \wedge \dots \wedge (A_n\sigma \leftarrow)$$

となることに注意されたい。

命題 2.2 P を確定節プログラム、 $C = A_0 \leftarrow A_1, \dots, A_n$ を確定節とする。プログラム $P \wedge \neg(C\sigma^-)$ におけるゴール $\neg(C\sigma^+)$ の SLD 反駁が存在するとき、 $P \models C$ である。

証明 (概要) $C = A_0 \leftarrow A_1, \dots, A_n$ が基礎である確定節であるとする。 $P \wedge \neg(C\sigma^-)$ は P に基礎単位節を追加したプログラム $P \wedge (A_1 \leftarrow) \wedge \dots \wedge (A_n \leftarrow)$ である。一方 $\neg(C\sigma^+)$ は $\text{ゴール} \leftarrow A_0$ である。 $P \wedge (A_1 \leftarrow) \wedge \dots \wedge (A_n \leftarrow)$ において $\text{ゴール} \leftarrow A_0$ からの SLD 反駁が成功するとき、 A_0 は $P \wedge (A_1 \leftarrow) \wedge \dots \wedge (A_n \leftarrow)$ のすべてのエルブランモデル I の要素である。

任意の P のモデル I に対して、 I が $(A_1 \leftarrow) \wedge \dots \wedge (A_n \leftarrow)$ のモデルであるならば、 I は $P \wedge (A_1 \leftarrow) \wedge \dots \wedge (A_n \leftarrow)$ のモデルであり、 I は A_0 のモデルとなる。 I は C のモデルである。 I が $(A_1 \leftarrow) \wedge \dots \wedge (A_n \leftarrow)$ のモデルでないときも、 I において C の仮定部分が偽となるので I は C のモデルとなる。よって $P \models C$ である。

C に変数が含まれるときも、変数をスコールム化することによって、同様に証明することができる。

例 2.4 次の確定節プログラム P と確定節 $C = p(f(a)) \leftarrow r(a)$ を考える。

$$P = \left\{ \begin{array}{l} p(f(X)) \leftarrow q(X) \\ q(X) \leftarrow r(X) \end{array} \right\}$$

2.4 含意の証明

すると、 $\neg(C\sigma^-)$ は基礎単位節 $r(a) \leftarrow$ であり、 $\neg(C\sigma^+)$ は $\text{ゴール} \leftarrow p(f(a))$ である。 $\text{ゴール} \leftarrow p(f(a))$ の

$$P \wedge \neg(C\sigma^-) = \left\{ \begin{array}{l} p(f(X)) \leftarrow q(X) \\ q(X) \leftarrow r(X) \\ r(a) \leftarrow \end{array} \right\}$$

における SLD 反駁は成功するので $P \models C$ である。

第 3 章

帰納論理プログラミング

帰納推論とは、与えられたデータから、それを説明する一般的な規則を導き出す推論である。こうした推論は、種々の学習假定や人間の言語習得の過程、データ集合の持つ特徴抽出、入出力例からのプログラムの自動合成、エキスパートシステムにおける知識の半自動獲得、データベースからの新知識の発見などに応用できる。本章では、こうした応用面を持ちうる帰納推論についての基礎的な定義について述べた後、論理プログラムを用いた帰納推論である帰納論理プログラミングの形式的な定義を与える。

3.1 帰納推論の基礎的概念

学習する主体を学習者 (learner) と呼ぶ。逆に、学習者にデータを与えるものを教師 (teacher) と呼ぶ。例えば、人間の子供が母国語を習得する際にはその子の親やまわりの人間が教師であり、その子供が学習者である。科学者がある自然現象の実験データから、ある法則を発見する場合は、自然現象が教師であり科学者が学習者となる。機械学習 (machine learning) とは、学習者が学習アルゴリズムや学習機械であるときの学習をいう。帰納推論 (inductive inference) 帰納推論とは、与えられたデータから、それを説明する一般的な規則を導き出す推論の総称である。

例えば、いま推論の対象が変数 x の 2 次の多項式であることだけは知っているものとする。変数 x と未知関数 $f(x)$ の値の対の列が、 $(0, f(0)), (1, f(1)), (2, f(2)), \dots$ の形でデータとして次々に与えられるとき、 $f(x)$ を帰納推論することを考える。まず $(0, 1)$ が与えられたとき、自然な仮説として $f(x) = 1$ が考えられる。次に $(1, 1)$ が与えられたとき、直前の推測 $f(x) = 1$ は満たされるので、この時点では推測を変更する必要はない。次に $(2, 3)$ が与えられたとき、上の推測では具合が悪いので、それを捨てて何回かの試行錯誤の後に、それまで

に与えられた事実 $(0, 1)$, $(1, 1)$, $(2, 3)$ のすべてを満たすような新しい推測 $f(x) = x^2 - x + 1$ を推測する。以後, $(3, 7)$, $(4, 13)$, $(5, 21)$, ... に対しては, すべて $f(x) = x^2 - x + 1$ が満たされるので, その推測を変更する必要はない。このようにして, 変数と関数の値の対の列 $(0, 1)$, $(1, 1)$, $(2, 3)$, $(3, 7)$, ... から, 2次の多項式 $f(x) = x^2 - x + 1$ を帰納推論することができる。

上の例では, x の値が $1, 2, 3, \dots$ である $(x, f(x))$ のようなデータが, 1個ずつ順序正しく学習者に与えられているが, 順序正しくデータが与えられない帰納推論も考えられる。また, 各時点における推測はそれまでに与えられたすべてのデータを説明している。しかし, もちろんその次に与えられるデータがこの推測を満たすとは限らない。上の場合は2次の多項式を仮定しているので, 異なる3つのデータが与えられればその規則を同定することができるが, 一般の場合には推論が無限に続くこともある。

帰納推論を正確に定義するには, 次の五つの項目を規定する必要がある。

1. 推論の対象になる規則の集合
2. 規則の表現方法
3. 例の提示 (例示) のしかた
4. 推論の方法
5. 推論を正しいと見る基準

以下でそれぞれの項目について述べる。

3.1.1 推論の対象になる規則の集合

先の例では, 対象になる規則の集合を x の2次関数としたが, 帰納推論において対象になる規則の集合を決めておくことが一般的である。これは, 帰納推論の可能性, 不可能性, 計算量を計るときに必要となる。推論の対象となる規則としては, 帰納的関数や形式言語, プログラムなどが考えらる。

3.1.2 規則の表現方法

規則の表現としては, 関数を計算する Turing 機械や言語を規定する文法, 論理プログラムなどが考えられてきた。Turing 機械は計算機の数学的モデルであり, それはプログラム

と見なすことができる。形式言語を対象とする場合には, 規則としての言語, その表現として文法を扱うので, 文法推論と呼ばれている。また, 本論文で扱う帰納論理プログラミングは規則の表現方法が一階述語論理, 特に論理プログラムである場合の枠組みである。

関数 f に対する例示には, 先に示した例のように入出力の対 $(x, f(x))$ の列や, f を計算する Turing 機械の動作系列などが考えられる。推論機械に関数 f の入出力の対 $(x, f(x))$ を与えることは, プログラムの自動合成システムに, 入力 x と f を計算するプログラムの入力 x に対する出力 $f(x)$ とを与えることに対応する。この意味で, 例からのプログラムの自動合成は関数の帰納推論とみなすことができる。また, 推論の対象が集合である場合には集合に属するデータと属さないデータの2種類データが考えられる。前者を正データ, 後者を負データと呼ぶ。文脈によっては, これらをそれぞれ正例と負例, 正の訓練例と負の訓練例, 正の観測例と負の観測例, 正の具体例と負の具体例と呼ぶこともある。たとえば, 形式言語は語の (無限) 集合であるから, 言語 L の正例は L に属する語であり, 負例は属さない語である。帰納推論の対象を集合とするときは, 集合を概念とみなすことができるので概念学習 (concept learning) を呼ばれている。

推論の対象がプログラムの場合には, プログラミング言語のパラダイムに依存する。Lisp などの関数型プログラムについては関数の場合と同様に考えられるが, 論理プログラムについては, 事実 (fact) の形, タプル (tuple) の形, または含意 (entailment) の形で例示する。たとえば, 事実とその真偽値の組 $(father(a, b), true), (mother(b, c), false), (father(c, d), true)$ といった形である。推論の対象とする述語は関係を定義しているとみなすことができるので, $(\langle a, b \rangle, true), (\langle c, d \rangle, false)$ のようにタプルと真偽値の組であって構わない。また, 論理プログラムは含意によってもの意味付けされるので,

$$(grandfather(a, c) \leftarrow father(a, b), mother(b, c), true)$$

のように含意の形で例示する場合もある。このような場合は特に含意からの学習 (learning from entailment) と呼ばれている。これらの例示は, 言語でいうと $true$ と付値されたデータが正データ, $false$ と付値されたデータが負データに対応する。

帰納推論の対象が集合が無限集合である場合や, 対象が無限濃度の定義域を持つ関数の場合にはすべての例を一度に与えることはできない。そこで, 例を次々に一つずつ与えてゆく場合や, 例の部分集合を一度に与える場合が考えられる。先に挙げた2次関数の例では, 教師が一方的に学習者に例を与えているが, 学習者が教師に対して積極的に質問をし, それに教師が答えるという場合も考えられる。先の例の場合であると, ある x の値に対し

て $f(x)$ の値を教師に質問するなどが考えられる。教師がある分野の専門家であれば、学習機械が様々なケースを分類するために、その専門家に質問することは妥当なことであろう。Angluin は概念学習における質問として、所属性質問、等価性質問、包含性質問、非交差性質問、全体性質問などを挙げている [Angluin 88]。

所属性質問 (membership query) とは、質問された要素が未知の概念に入っているかどうかという 1 ビットの情報を求める質問である。等価性質問 (equivalence query) とは、推測した規則を教師に提示しその規則が未知の概念を正しく表しているかどうかを問う質問である。推測した規則が間違っている場合には、反例つまり、未知の概念と推測した規則を表す概念の差の要素が学習者に与えられる。所属性質問は、学習者が効率的な学習を達成するのに重要な役割を演じる。所属性質問と等価性質問に答える能力を持った教師は、**極小最適教師** (minimum adequate teacher) と呼ばれている。極小最適教師は、学習機械にとって最低限必要であり、最も妥当であると考えられる能力を持った教師として提案された。計算機上に半自動的にエキスパートシステムを構築することを考えたとき、この考え方は自然であろう。たとえば、ある分野の専門家、たとえば、癌の専門家に計算機が質問を繰り返すことで、癌診断のエキスパートシステムを作るといった具合である。

例を次々に一つずつ与えてゆく場合には推論は

入力要求 → 推測の計算 → 出力

という基本過程を無限に繰り返すことによって行うものとする。つまり、推論は、次々に例を入力として受取り、その時点での推測を計算し、その結果を出力する。各時点における推測は、それまでに受け取った例の有限列に基づいていると考えられるので、このことは、例えば人間の言語習得の過程やソフトウェアの改良に伴う進化などを考えれば不自然なことではない。また、例の部分集合を一度に与える場合には推論は上の基本過程が一回だけ行われるものである。

3.1.3 推論を正しいと見る基準

推論を正しいとみなす基準としては、帰納推論が前述のように無限に続く過程であるということを念頭に置いた極限における同定や、ある有限の過程で必ず正しく帰納推論が完了するという EXACT 学習や、ほとんどの確率でほとんどのほどほどに正しく学習ができるという PAC 学習モデルなどが一般的である。

極限における同定 (identification in the limit) は Gold によるもので、推論機械 M が規則 R を極限において同定するとは、 R の例示が与えられたときに M の生成する出力の列がある表現 τ に収束し、すなわちある時点以降の M の出力がすべて τ であり、しかも τ が R の正しい表現になっていることをいう。また、推論機械 M が規則 R を **EXACT 学習** (EXACT learning) するとは、 R の例示が与えられたときに、 R や与えられた例の複雑さをあらわすパラメータによる有限ステップ以内に M の R の正しい表現である τ を必ず出力することをいう。**PAC 学習** (PAC learning) については、どんな R もどんな確率分布の元で R の例が与えられたとしても、信頼性の高い確率で、信頼性の高い解を見つけるような推論を正しく見る基準である。

3.1.4 推論の方法

帰納推論を実際の計算機で実現する際には推論の方法が重要となる。推論の方法には、大きく分けて**枚挙的手法** (enumerative method) と**構成的手法** (constructive method) の2通りがある。**枚挙的手法**とは、すべての可能な仮説を枚挙して、それまでに受け取ったデータを説明できるかどうかをテストすることによって正しい推測を発見するという手法である。この手法は、一般には生成テスト法 (generate and test) と呼ばれており、現在の計算機のもっとも得意とする繰り返しの用いた手法である。

枚挙手法を用いて先の例の仮説を生成することを考えてみる。まず、未知の関数を $f(x) = ax^2 + bx + c$ とおき、定数 a, b, c の組をたとえば、次のような順序で仮説を枚挙ゆく。

$$\begin{aligned}
 (a, b, c) &= (0, 0, 0) & \cdots & f(x) = 0 \\
 (a, b, c) &= (0, 0, 1) & \cdots & f(x) = 1 \\
 (a, b, c) &= (0, 1, 0) & \cdots & f(x) = x \\
 (a, b, c) &= (1, 0, 0) & \cdots & f(x) = x^2 \\
 (a, b, c) &= (0, 0, -1) & \cdots & f(x) = -1 \\
 (a, b, c) &= (0, -1, 0) & \cdots & f(x) = -x \\
 (a, b, c) &= (-1, 0, 0) & \cdots & f(x) = -x^2 \\
 (a, b, c) &= (0, 1, 1) & \cdots & f(x) = x + 1 \\
 (a, b, c) &= (0, 1, -1) & \cdots & f(x) = x - 1 \\
 (a, b, c) &= (0, -1, 1) & \cdots & f(x) = -x + 1 \\
 (a, b, c) &= (0, -1, -1) & \cdots & f(x) = -x - 1
 \end{aligned}$$

$$\begin{aligned}
(a, b, c) &= (1, 0, 1) & \cdots & f(x) = x^2 + 1 \\
(a, b, c) &= (1, 0, -1) & \cdots & f(x) = x^2 - 1 \\
(a, b, c) &= (-1, 0, 1) & \cdots & f(x) = -x^2 + 1 \\
(a, b, c) &= (-1, 0, -1) & \cdots & f(x) = -x^2 - 1 \\
(a, b, c) &= (1, 1, 0) & \cdots & f(x) = x^2 + x \\
(a, b, c) &= (1, -1, 0) & \cdots & f(x) = x^2 - x \\
(a, b, c) &= (-1, 1, 0) & \cdots & f(x) = -x^2 + x \\
(a, b, c) &= (-1, -1, 0) & \cdots & f(x) = -x^2 - x \\
(a, b, c) &= (0, 0, 2) & \cdots & f(x) = 2 \\
(a, b, c) &= (0, 0, -2) & \cdots & f(x) = -2 \\
(a, b, c) &= (0, 2, 0) & \cdots & f(x) = 2x \\
(a, b, c) &= (0, -2, 0) & \cdots & f(x) = -2x \\
&\vdots
\end{aligned}$$

これらの仮説一つ一つに対して、これまで与えられた例に矛盾しないかどうかをテストしてゆき、何十回かの生成とテストの後、 $(a, b, c) = (1, -1, 1)$ で正しい仮説 $f(x) = x^2 - x + 1$ を得る。Shapiro のモデル推論システム [Shapiro 81] も原理的には枚挙手法に基づいており、一階述語論理の性質を利用することによりこの戦略を高速化したものである。

一方、**構成的手法**とは、対象とする規則で成り立つ原理を用いて、例を説明する仮説を導き出す手法である。いま、対象としている規則は x の2次関数であるので、枚挙を用いなくても互いに異なる三つの例が与えられた時点でこの関数を同定することが原理的に可能であることは明らかであろう。つまり、 $f(x) = ax^2 + bx + c$ とおき $f(0) = 1, f(1) = 1, f(2) = 3$ から連立方程式

$$\begin{aligned}
c &= 1 \\
a + b + c &= 1 \\
4a + 2b + c &= 3
\end{aligned}$$

をたて、この連立方程式を解けばよい。構成的な帰納推論を実現する上で重要なのは対象とする規則で成り立つ原理である。

3.2 帰納論理プログラミング

帰納論理プログラミング (Inductive Logic Programming, ILP) とは機械学習や知識発見を論理プログラミングの枠組みを用いて行う研究領域である。以下では、ILP の形式的な定義を ILP が抱える現在の問題点に注意しながら述べる。

ILP 理論の目標は、与えられた多くの例からそれを説明する節形式文を出力する帰納学習システムを構築することである。形式的には、与えられた正の具体例 $E_1^+, \dots, E_n^+$ と負の具体例 $E_1^-, \dots, E_m^-$ に対して条件 1.1 を満たす論理プログラム P を計算するというものである。これは、論理プログラミングが、与えられたプログラム P に対して、

$$P \vdash A_1 \wedge \dots \wedge A_n$$

となる原子式 $A_1 \wedge \dots \wedge A_n$ を計算することを目的としていることにちょうど逆になる。

あらかじめ学習者に論理プログラムで表された背景知識 B が与えられている場合には条件 1.2 を満たす節 H を計算する。

最初に「例」と「仮説」について、Shapiro モデル推論システム [Shapiro 81] を例に考察してみる。いま、システムにリストの反転プログラム P を学習させたいとする。モデル推論システムでは、例と P の最小エルブランモデル $M(P)$ の要素である基礎原子式を正の例、そうでないものを負の例として扱う。たとえば、正例として

$$\text{rev}([1, 2], [2, 1])$$

負例として

$$\text{rev}([1, 2], [1, 2])$$

などをシステムに次々と与えてゆく。システムはその例から仮説 P' を生成する。目標は、 $M(P')$ が $M(P)$ に一致するか、もしくは十分近いようなプログラム P' を得ることである。この場合であると、システムが多くの例を受け取った後、たとえば

$$P' = \left\{ \begin{array}{l} \text{rev}([], []) \leftarrow \\ \text{rev}([A|X], Z) \leftarrow \text{rev}(X, Y), \text{append}(X, [A], Z) \\ \text{append}([], X, X) \leftarrow \\ \text{append}([A|X], Y, [A|Z]) \leftarrow \text{append}(X, Y, Z) \end{array} \right\}$$

のようなプログラムを出力すれば正しい推論が行われたことになる。この際、モデル推論

システムでは、学習の対象となる概念は述語 rev があらかず関係つまり、タプルの集合

$$\left\{ \begin{array}{l} \langle [], [] \rangle, \langle [a], [a] \rangle, \langle [b], [b] \rangle, \langle [c], [c] \rangle, \dots \\ \langle [a, b], [b, a] \rangle, \dots, \langle [a, b, c], [c, b, a] \rangle, \dots \end{array} \right\}$$

であるのに対して、教師は

$$append([c, b], [a], [c, b, a])$$

のように補助述語 $append$ の具体例も与えなければならないことに注意しておく。

3.3 補助述語

論理プログラミングにおいてこれまでも問題になっていたのが補助述語の扱いである。補助述語 (auxiliary predicate) とは学習の対象となる関係 R を表現するプログラムにおいて、 R を定義する述語以外の述語のことである。上の例では学習の対象となる関係 R が述語 rev で定義される関係であり、述語 $append$ が補助述語である。

ILP においては、学習の対象 R が補助述語 $aux_1, \dots, aux_n$ を用いて表現される場合、 $aux_1, \dots, aux_n$ も学習するのか、あるいは、これらは背景知識としてあらかじめ定義を与えておくのか、という学習における補助述語の扱い方を定めなければならない。Shapiro のモデル推論システム (MIS)[Shapiro 81] では、矛盾点追跡アルゴリズム中で補助述語に対する質問を用いてこの問題を回避している。Muggleton の Progol[Muggleton 94, Muggleton 95] では、補助述語に背景知識としてあらかじめ学習システムに与えられたものを使っている。

Kleene の定理によると制約を与えなければどのような概念も補助述語を用いた一階述語論理によって表現できる。

定理 3.1 [Kleene 52] 任意の帰納的に可算な一階言語 L 中の式の集合 C は L に有限個の述語記号を導入するした一階言語 L' によって、有限公理化できる。

関係 R を論理プログラムで表現するときに、補助述語なしのプログラムでは表現できないが、補助述語を用いたプログラムでは表現できる場合がある [Stahl 95]。以下にその例を示す。

例 3.1 ([Stahl 95]) 数を $s(s(s(0)))$ のように 0 の successor を用いて表現し、述語 $square(X)$ を平方数を表す述語とする。すると、平方数の概念は 1 引数の述語 $square$ と 1 引数の関数記号 $s/1$ と定数 0 だけを仮説言語としたとき平方数を表すプログラムは存在しないが、いくつかの補助述語を用いたときには平方数を表すプログラムが存在する。

証明

解 P が存在するとする。 P 中の節は $square(s^{n^2}(0))$ か $square(s^m(X)) \leftarrow Body$, という形をしていて $Body$ には定数記号や頭部に現れない変数はないと仮定することができる。なぜなら、リテラル $square(s^k(Z))$, $Z \neq X$, $square(s^k(0))$ は、常に真か偽のどちらかであるからである。つまり、 P 中の各節 C は

$$square(s^m(X)) \leftarrow square(s^{k_1}(X)), \dots, square(s^{k_l}(X))$$

のような形をしていて $k_j \neq m$ である。節 C が $V = s^v(0)$, $v \geq m$ である数で呼ばれたとする。すると、 $k_j > m$ は V にある定数 c_j を足して $square$ を呼び出していると考えることができる。同様に $k_j < m$ は V から定数 c_j を引いて $square$ を呼び出していると考えることができる。したがって、 C は非形式的には次のように書くことができる。

$$square(V) \leftarrow square(V \pm c_1), \dots, square(V \pm c_l), c_j > 0.$$

c を P 中の非単位節の最大の c_j とし、 n_{max} を単位節 $square(s^n(0))$ の最大の n とする。 w を $w > \max\{n_{max}, c\} + 1$ とし、以下の SLD 導出木によって $P \vdash square(w^2)$ となると仮定する。

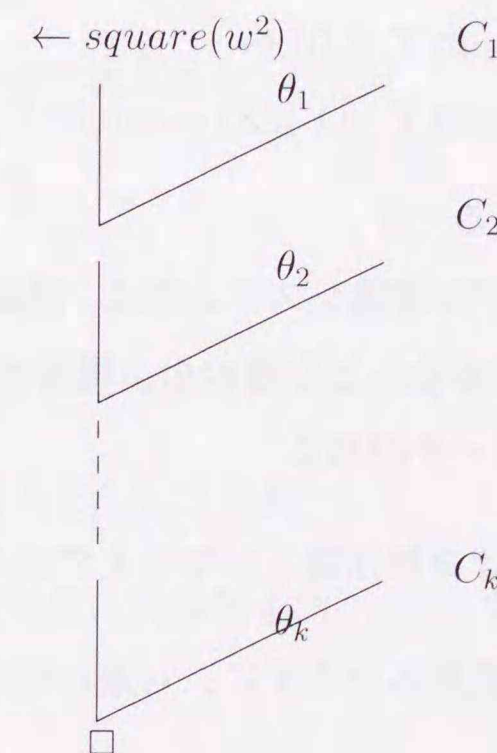


図 3.1: $\leftarrow square(w^2)$ からの SLD 反駁

ただし、 $C_1, \dots, C_k \in P$ である。SLD 導出の線形性より、これは次のように書くことが

できる.

$$(\dots (\leftarrow \text{square}(w^2) \cdot C_1 \theta_1) \cdot) \cdot C_k \theta_k.$$

ここで, $(A \cdot C\theta)$ はゴール A と確定節 C に代入 θ による導出の結果得られたゴールを意味する. $w > n_{\max}$ より, $C_1 \theta_1$ はつぎのような形をしていなければならない.

$$\text{square}(w^2) \leftarrow \text{square}(w^2 \pm c_1), \dots, \text{square}(w^2 \pm c_l), c_j > 0.$$

この導出が成功するためには, 各リテラル $\text{square}(w^2 \pm c_j)$ が P から導出されなければならない. しかし, 各 c_j に対して

$$(w-1)^2 < w^2 - c_j < w^2 < w^2 + c_j < (w+1)^2$$

が成り立つので, ある非平方数 u に対して $P \vdash \text{square}(u)$ となってしまう, P がこの問題の解であるという仮定に矛盾する.

しかし, 掛け算を表す $\text{mult}/3$ と 足し算を表す $\text{add}/3$ をもちいると, 平方数は

$$\begin{aligned} \text{square}(X) &\leftarrow \text{mult}(Z, Z, X). \\ \text{add}(X, 0, X) &\leftarrow \\ \text{add}(X, s(Y), s(Z)) &\leftarrow \text{add}(X, Y, Z) \\ \text{mult}(X, 0, 0) &\leftarrow \\ \text{mult}(X, s(Y), Z) &\leftarrow \text{mult}(X, Y, W), \text{add}(W, X, Z) \end{aligned}$$

の様に表すことができる. $\square$

先に, Shapiro のモデル推論システムでは, 補助述語に対する例を与える方法をとっている. これは少々不自然であることを述べた. 現在の ILP 研究では, 補助述語の扱いに対して主に次の二つのアプローチがある.

1. 補助述語の定義は背景知識として与えておく.
2. 補助述語の例は直接あたえずに, 対象の例から自動的に作り出す.

3.4 背景知識を用いた構成的な仮説の生成問題

補助述語の定義を背景知識として与えておく方法は, 現在 ILP へのシステムでの主流である. Muggleton の Progol[Muggleton 94, Muggleton 95] や FOIL[Quinlan 96] などの代表

的な ILP システムはこの手法を採用している. 3.2 節の例を用いると背景知識としてあらかじめ,

$$B = \left\{ \begin{array}{l} \text{append}([], X, X) \leftarrow \\ \text{append}([A|X], Y, [A|Z]) \leftarrow \text{append}(X, Y, Z) \end{array} \right\}$$

というプログラムを与えておき, 節

$$\begin{aligned} H_1 &: \text{rev}([], []) \leftarrow \\ H_2 &: \text{rev}([A|X], Z) \leftarrow \text{rev}(X, Y), \text{append}(X, [A], Z) \end{aligned}$$

を帰納推論する. ここで, 注意すべきことは, H_1 と H_2 は別の過程で生成されることである. たとえば, はじめに背景知識 B のもとで, 例

$$\text{rev}([], [])$$

から H_1 を帰納推論し, つぎに例

$$\text{rev}([a], [a])$$

から, 背景知識 $B \wedge H_1$ から H_2 を帰納推論する. 従来の帰納論理プログラミング手法の問題点として,

1. 例と背景知識の区別がないこと
2. 与えられる例の順番に依存する手法であること
3. 扱える規則に大きな制限があること

が挙げられる.

本論文では, 背景知識 B と与えられた正の具体例 $E_1^\oplus, \dots, E_n^\oplus$ に対して,

$$B \wedge H \vdash E_1^\oplus \wedge \dots \wedge E_n^\oplus$$

となる仮説 H を背景知識 B を用いて相対的に具体例 $E_1^\oplus, \dots, E_n^\oplus$ を一般化することによって構成的に生成する手法を示す.

3.5 質問を用いた補助述語発見問題

一階述語論理の枠組みでは仮説空間の要素は一般に無限個にある. 推論を実現しやすくするためには何らかの形で仮説空間を制限する必要がある. このとき制限された仮説空間

が無限であっても構わない。この制限はバイアス (bias) と呼ばれ、仮説の語彙、つまり、述語記号、関数記号、構文を制限する。

もし、学習の対象としているの概念が現在のバイアスで表すことができない場合、学習アルゴリズムは言語のバイアスを変更しなければならない。このようなバイアスの変更は、バイアスのシフト (bias shift) と呼ばれている。言語のバイアスをシフトするには、言語の制限を緩める、または、新しい述語を導入するという二つの方法がある。つまり、先の *rev* の例を用いると述語 *rev* だけを用いて述語 *rev* だけを帰納推論をすすめて、どうしても *rev* だけでは帰納推論できないと判断したときに新たな適当な述語たとえば aux_1 を導入する枠組みである。 aux_1 の定義が *append* の定義に一致していることが理想である。この方法を述語発見 (predicate invention, PI) という。

述語発見は困難な問題であり、様々なアプローチから研究が行われている。補助述語を効率よく発見するための一つのアプローチとして、あらかじめプログラムのスキーマを与える手法が提案されている [Stahl 95]。本論文では、CIA[Raedt&Bruynooghe 92] や RDT[Wirth&O'Roke 91] と同様に二階の論理における述語変数を用いた節の形で表す。例えば、 P_0, P_1 を述語変数とした節形式文

$$\Sigma = \left\{ \begin{array}{l} P_0(f(X)) \leftarrow P_1(X) \\ P_0(g(X)) \leftarrow P_1(X) \end{array} \right\}$$

をスキーマとし、 P_0, P_1 に述語記号を代入した節を対象の推論に用いるのである。しかし、プログラムのスキーマに関数記号の出現を許すと、たとえスキーマを使っても効率的な学習は不可能である。なぜなら、スキーマを用いて正則言語のクラスは表現でき、Gold の定理 [Gold 78] により、正則言語のクラスは与えられた正負の例からでは効率的な学習ができないからである。つまり、背景知識を用いない ILP の枠組み

$$\begin{array}{l} P \vdash E_1^{\oplus} \wedge \dots \wedge E_n^{\oplus} \\ P \not\vdash E_1^{\ominus} \vee \dots \vee E_n^{\ominus} \end{array}$$

において、この条件を満たす論理プログラム P を計算するという枠組みでは述語発見を用いた場合効率よい帰納推論は不可能である。

一方、Angluin による形式言語の帰納推論の研究結果 [Angluin 87] により、正則言語は質問を用いることにより効率的な帰納推論が可能となることが知られている。また、Sakakibara は Angluin の学習アルゴリズムを拡張し、木オートマトンを質問によって帰納推論するアルゴリズムを与えている [Sakakibara 90]。そこで、これらの学習アルゴリズムがどのよう

な論理的性質を用いているのかを明らかにし、帰納論理プログラミングにおける一般的手法を導く。学習者が補助述語を自動生成するということもこの点に結び付く。

3章のまとめ

本章では、帰納推論に関する基礎的概念と帰納論理プログラミングという枠組みについて概説した。また、従来の帰納論理プログラミング手法の問題点を述べた。

第 4 章

論理プログラムを用いたアブダクションの実現

本章では、論理プログラミングに対するアブダクションの手法である SOLDR 導出を提案する。SOLDR 導出は SOLDR 融合 [Yamamoto 97a, Yamamoto 98c] に失敗によるスキップという考え方を導入した手続きである。

はじめに、アブダクションとはどのような推論であるかを述べ、そのあと SOLDR 導出を提案する。また、SOLDR 導出を構成的な帰納推論に応用するために SOLDR 導出を用いたアブダクションのプロセスの停止性問題について述べる。最後に SOLDR 導出を Prolog インタプリタ上に SOLDR インタプリタとして実装する。

4.1 論理プログラムを用いた基礎事実のアブダクション

アブダクション (abduction, 発想推論) は、ある事実 E が観測されたときに、現在持っている理論 P を用いて $P \wedge A \vdash E$ となる事実を見つける手法である。アブダクションは演繹推論と深く関係し、その歴史は哲学にさかのぼる [Peirce]。たとえば、演繹推論は「ソクラテスは人間である」と「すべての人間はいつか死ぬ」から「ソクラテスは死ぬ」を導く推論であるが、発想推論は「すべての人間はいつか死ぬ」と「ソクラテスは死んだ」から「ソクラテスは人間である」を導く推論である。

論理プログラムに対するアブダクションを以下のように定義する

定義 $P \vdash E$ である基礎事実 E と、論理プログラム P に対して

$$P \wedge A \vdash E$$

となる基礎事実 A を求めることをアブダクションという。このとき、 A は E の P に関するアブダクションの帰結という。

いか，論理プログラム P に対するアブダクションの例を挙げる．

例 4.1 つぎのような論理プログラム P を考える．

$$P = \left\{ \begin{array}{l} \text{mother}(\text{sazae}, \text{tara}) \leftarrow \\ \text{father}(\text{norisuke}, \text{tara}) \leftarrow \\ \text{father}(\text{namihei}, \text{katsuo}) \leftarrow \\ \text{father}(\text{namihei}, \text{wakame}) \leftarrow \\ \text{grandfather}(X, Z) \leftarrow \text{father}(X, Y), \text{father}(Y, Z) \\ \text{grandfather}(X, Z) \leftarrow \text{father}(X, Y), \text{mother}(Y, Z) \end{array} \right\}$$

このプログラム P は，ある家族の親子関係を表す不完全なプログラムである．すなわち， P に入るべきある基礎事実が入っていないため，次の基礎事実 E を導出することができない．

$$E = \text{grandfather}(\text{namihei}, \text{tara}) \leftarrow$$

つまり， $P \not\models E$ である．いま， E の P に関するアブダクションの帰結 A には

$$\begin{aligned} A_1 &= \text{father}(\text{katsuo}, \text{tara}) \leftarrow \\ A_2 &= \text{father}(\text{wakame}, \text{tara}) \leftarrow \\ A_3 &= \text{father}(\text{namihei}, \text{norisuke}) \leftarrow \\ A_4 &= \text{mother}(\text{katsuo}, \text{tara}) \leftarrow \\ A_5 &= \text{mother}(\text{wakame}, \text{tara}) \leftarrow \\ A_6 &= \text{father}(\text{namihei}, \text{sazae}) \leftarrow \\ A_7 &= \text{grandfather}(\text{namihei}, \text{tara}) \leftarrow \end{aligned}$$

実際，上の各事実 A_i に対して，ゴール $\leftarrow \text{father}(\text{namihei}, \text{tara})$ からの SLD 反駁は成功し， $P \wedge A_i \vdash E$ となる．

4.2 枚挙と SLD 反駁を用いた自明なアブダクションアルゴリズム

枚挙と SLD 反駁を用いた自明なアブダクションアルゴリズムとしては，図 4.1 のようなアルゴリズムが考えられる．

基本的には，この枚挙的なアルゴリズムで論理プログラムに関するアブダクションを求めることができるが，ゴール $\leftarrow A$ からの SLD 導出が無限の導出木を持つ可能性がある場合には，ステップ 3 は停止しないし， P または E に関数記号が含まれる場合には，いくらでも新しい基礎原子式が作ることができるのでステップ 7 におけるループは停止しない．

Input: 基礎原子式 E ．

Given: 確定節プログラム P ．

Output: $P \wedge A \vdash E$ となる基礎原子式 A ．

Procedure Enumerative Abduction:

- 1 $S = \phi$ とする；
 - 2 P と E で用いられている言語から集合 S に含まれない基礎原子式 A を作る；
 - 3 if ゴール $\leftarrow A$ からの SLD 反駁が成功する then
 - 4 A を出力する；
 - 5 endif
 - 6 集合 S に A を追加する；
 - 7 集合 S に含まれない基礎原子式 A が作れなくなるまで 2 から 6 を繰り返す；
- end .

図 4.1: 枚挙と SLD 反駁を用いた自明なアブダクションアルゴリズム

4.3 SOLDR 導出を用いた構成的なアブダクション手法

本節では，アブダクションの帰結を構成的に求める手法を提案する．それは SOLDR 導出と呼ばれる手法である．SOLDR 導出は基本的に SOLDR 融合 [Yamamoto 98c] と等しいが，若干の修正が加えられている．SOLDR 導出は，SOLDR 融合に失敗によるスキップという考え方を導入した手法である．

定義 P を確定節プログラム， G をゴール節， R を計算規則とする．4 つ組 $(G_i, L_i, \theta_i, C_i)$ ($i = 0, 1, \dots, n$) の有限列は，以下の条件を満たすとき (P, G) の SOLDR 導出という．

1. G_i と L_i はゴール節， θ_i は代入， C_i は P 中の節の変種であり， $i \neq j$ ならば C_i と C_j は変数を共有しない．
2. $G_0 = G$ (ただし， P における G の SLD 反駁は成功しない)， $L_0 = \square$ ($\square$ は空節)．
3. 任意の i について， $G_i \leftarrow A_1, \dots, A_m$ ， $L_i = \square$ または $\leftarrow B_i$ で， A_j が計算規則 R によって選択された原子論理式であるとき，次のいずれかが成立する．

(a) (融合)

θ_i は C_i^+ と $A_j \leftarrow$ の最汎単一化子であり,

$$G_{i+1} = \leftarrow (A_1, \dots, A_{j-1}, A'_1, \dots, A'_k, A_{j+1}, \dots, A_m) \theta_i,$$

$$C_i^- = \leftarrow A'_1, \dots, A'_k, L_{i+1} = L_i \theta_i.$$

(b) (スキップ1・帰結の追加)

$$L_i = \square \text{ のとき, } G_{i+1} = \leftarrow A_1, \dots, A_{j-1}, A_{j+1}, \dots, A_m, L_{i+1} = \leftarrow A_j.$$

(c) (スキップ2・帰結の単一化)

$$L_i = \leftarrow B_i \text{ のとき, } \theta_i \text{ は } A_j \text{ と } B_i \text{ の最汎単一化子であり,}$$

$$G_{i+1} = (\leftarrow A_1, \dots, A_{j-1}, A_{j+1}, \dots, A_m) \theta_i, L_{i+1} = \leftarrow B_i \theta_i.$$

$G_n = \square$ であるときのゴール節 L_n をこの SOLDR 導出の帰結という.

スキップ操作が全く用いられない SOLDR 導出は SLD 反駁と同一視することができる.
ここで, ゴールの集合 $SOLDR(P, G)$ を

$$\{\leftarrow A\theta \mid \leftarrow A \text{ は } (P, G) \text{ の SOLDR 導出の帰結, } \theta \text{ は } A \text{ を基礎にする代入}\}$$

と定義する.

SOLDR 融合は次の定理での特徴付けられている.

定理 4.1 [Yamamoto 98c] $P \wedge G$ が無矛盾であるとする. 任意の基礎原子式 A に対して, つぎの二つは同値である.

$$1. \leftarrow A \in SOLDR(P, G).$$

$$2. P \wedge G \models \neg A.$$

この定理は, SOLDR 導出によってゴール $G = \leftarrow E$ の P に関するアブダクションの帰結 A がすべて求まることを示している.

例 4.2 例 4.1 で用いた E からの P における SOLDR 導出を考える.

$$G_0 = \leftarrow grandfather(namihei, tara) \quad L_0 = \square$$

$$C_1 = grandfather(X, Z) \leftarrow father(X, Y), father(Y, X)$$

$$G_1 = \leftarrow father(namihei, Y), father(Y, tara) \quad L_0 = \square$$

$$C_2 = father(namihei, katsuo) \leftarrow$$

$$G_2 = \leftarrow father(katsuo, tara) \quad L_0 = \square$$

$$(father(katsuo, tara) \text{ をスキップ})$$

$$G_3 = \square \quad L_3 = \leftarrow father(katsuo, tara)$$

となり, 帰結 $A_1 = father(katsuo, tara)$ を得る. また,

$$G_0 = \leftarrow grandfather(namihei, tara) \quad L_0 = \square$$

$$C_1 = grandfather(X, Z) \leftarrow father(X, Y), father(Y, X)$$

$$G_1 = \leftarrow father(namihei, Y), father(Y, tara) \quad L_0 = \square$$

$$(father(namihei, Y) \text{ をスキップ})$$

$$G_2 = \leftarrow father(Y, tara) \quad L_2 = \leftarrow father(namihei, Y)$$

$$C_3 = father(norisuke, tara) \leftarrow$$

$$G_3 = \square \quad L_3 = \leftarrow father(namihei, norisuke)$$

により, A_3 を得る. 同様に SOLDR 導出により帰結 A_2, A_4, A_5, A_6, A_7 を得ることができ,

$$SOLDR(P, E) = \left\{ \begin{array}{l} \leftarrow father(katsuo, tara) \\ \leftarrow father(wakame, tara) \\ \leftarrow father(namihei, norisuke) \\ \leftarrow mother(katsuo, tara) \\ \leftarrow mother(wakame, tara) \\ \leftarrow father(namihei, sazae) \\ \leftarrow grandfather(namihei, tara) \end{array} \right\}$$

となる.

SOLDR 導出によって得られた帰結 L の基礎代入例を否定すると, アブダクションの帰結が得られる.

4.4 失敗によるスキップ

論理プログラミングでは $P \not\models A$ であるときに, $\neg A$ が成立するとみなす. これは失敗による否定と呼ばれている.

次の命題で, SOLDR 導出の帰結は論理プログラムの失敗による否定で特徴付けられる.

命題 4.1 P を確定節プログラム, E を基礎原子式とし, $P \not\models E$ を仮定する. すると, $\leftarrow A \in SOLDR(P, E)$ ならば, $P \not\models A$ である.

すなわち, P における E からの SOLDR 導出で得られた帰結すなわち, スキップされた原子式 A はもともと, P では説明できない原子式であり, この原子式が P に追加されると E が説明できるようになるようなものである.

系 4.1 P を確定節プログラム, E を基礎原子式とし, $P \not\models E$ を仮定する. $P \vdash A$ であるならば, $\leftarrow A \notin \text{SOLDR}(P, E)$ である.

この系は, SOLDR 導出の際, 選択された原子式に対して融合をスキップより優先させて実行し, 融合を優先させて失敗した場合にスキップを用いればよいことを示している. 一般に, 失敗による否定は第2章で与えた意味による否定には一致しない [Lloyd 84]. それは SLD 導出のステップが無限に続いてしまうことがあるからである. したがって, SOLDR 導出も無限に続いてしまう場合には帰結を導出できない.

4.5 SOLDR 導出の停止性問題とその解

SOLDR 導出によるアブダクションのプロセスの停止性を保証するために背景知識と仮説のクラスを制限する.

定義 確定節 $A_0 \leftarrow A_1, \dots, A_n$ が弱縮小節であるとは, 任意の代入 θ と任意の $1 \leq i \leq n$ に対して, $\|A_0\theta\| \geq \|A_i\theta\|$ であることをいう. ここで, $\|A_i\|$ は A_i 中に現れる関数記号, 定数記号, 変数の出現数の総和である.

P を弱縮小節からなるプログラムに制限すると, (P, G) からの SOLDR 導出の帰結は有限個であり, それらはすべて変数を含まない. よって, $\text{SOLDR}(P, G)$ は有限集合となる.

定理 4.2 P を弱縮小節からなるプログラムに限定すると, P における G からの SOLDR 導出は有限ステップで停止して必ず正しい答えを出力する.

4.6 SOLDR 導出の実装

SOLDR 導出は SLD 導出にスキップ操作を追加したものであるから, Prolog インタプリタを拡張することで SOLDR 導出を実現することができる.

SOLDR 導出の Prolog を用いた実装の主要部を 4.2 に示す. Prolog の詳細についてはテキスト等 [Clocksin&Mellish 94, Sterling&Shapiro 86] やマニュアル [SICStus 96] を参照されたい.

述語 `soldr/3` は基本的に Prolog メタインタプリタであるが, スキップ操作が追加されている. 確定節プログラム P とゴール節 G が与えられたとき, 手続き `soldr(G, P, A)` は $(\leftarrow A) \in \text{SOLDR}(P, G)$ となる原子式 A を計算する. 融合をスキップより上の位置で定義しているので, Prolog の失敗によるスキップが実現されている. Prolog のバックトラッ

```
soldr([], _Prog, []).
soldr([G|Gs], Prog, L):-
    copy_term(Prog, Prog2), member([G, :-|Bs], Prog2),
    append(Bs, Gs, NewG), soldr(NewG, Prog, L).
soldr([G|Gs], Prog, [G]):-
    soldr(Gs, Prog, []).
soldr([L|Gs], Prog, [L]):-
    soldr(Gs, Prog, [L]).
```

図 4.2: 底節交差汎化法を用いて仮説を求める Prolog プログラム

ク機構により, スキップする原子式が異なった複数の帰結を得る. `soldr/3` が用いる中間述語 `copy_term/2` は項の変種を計算する Prolog の一般的な組み込み述語であり, 中間述語 `append/3` は二つのリストの連結を求める述語, 中間述語 `member/2` は与えられたリストの要素を一つ求める述語である.

次に, SOLDR 導出の実行例を示す.

```
| ?- P=[ [father(namihei,katsuo),:-],
         [father(namihei,wakame),:-],
         [mother(sazae,tara),:-],
         [father(norisuke,tara),:-],
         [grandfather(X,Y),:-,father(X,Z),father(Z,Y)],
         [grandfather(X,Y),:-,father(X,Z),mother(Z,Y)] ],
soldr([grandfather(namihei,tara)],P, [A] ).
```

```
A = father(katsuo,tara) ? ;
A = father(wakame,tara) ? ;
A = father(namihei,norisuke) ? ;
A = mother(katsuo,tara) ? ;
A = mother(wakame,tara) ? ;
```



```

A = father(namihei,sazae) ? ;
A = grandfather(namihei,tara) ? ;
no

```

これは例4.1で用いたプログラムにおける事実 *grandfather(namihei,tara)* のアブダクションの帰結がすべて得られたことを示している。

4章のまとめ

本章では、論理プログラムに対するアブダクション手法である SOLD R 導出を与えた。そして、失敗によるスキップの考え方を導入し、これについて議論した。さらに、SOLD R 導出が必ず停止という特殊な条件を与えた。SOLD R 導出が SLD 導出の自然な拡張であることを用いて、Prolog インタープリタ上で SOLD R 導出を実装した。

第5章

複数の例に共通する規則の発見手法

本章では、底節交差汎化法という複数の具体例に共通する規則の発見手法を与える。底節交差汎化法とは、含意の形で与えられた各具体例の底節を組み合わせ、包摂に基づく最小汎化を計算し、最小汎化が確定節となる節を仮説を見つける学習手法である。また、同手法を論理プログラミング言語 Prolog を用いて ILP システム BORDA として実装する。

5.1 本章で扱う問題

本章では、含意の形で与えられた正の例 $E_1, \dots, E_n$ と背景知識 B に対して、

$$B \wedge H \vdash E_1 \wedge \dots \wedge E_n$$

となる節 H を構成的に計算する手法を扱う。

例として与えられた含意を説明する仮説の生成手法として、逆伴意法 [Muggleton 95]、飽和法 [Rouveirol 92]、 V^n 演算 [Jung 93]、底汎化法 [Yamamoto 97b] などが提案されている。中でも、底汎化法を用いた仮説の生成手法は

1. その健全性、完全性が示されている、
2. 仮説の生成 (インダクション) が発想推論 (アブダクション) と演繹推論 (デダクション) の組み合わせによって実現されている

という二つの特徴を持つ。しかし、底汎化法は理論的な結果であり、底汎化法に基づく具体的な帰納推論システムはこれまで実装されていなかった。その理由として以下の二つが挙げられる。

1. 底汎化法では、複数の具体例が与えられる場合の考慮がされていない,
2. 同手法によって生成される仮説は、いくらでも長くなる可能性がある.

本章では、底節交差汎化法という底汎化法の複数の具体例への拡張を与え、最初の問題を解決する. また、背景知識と仮説にある制限を課したクラスに対して、底節交差汎化法を適用することで二番目の問題を解決する.

5.2 底節交差汎化法の概要

本節では、底節交差汎化法の概要を述べる底節交差汎化法は Yamamoto による底汎化法を複数の例を扱えるように拡張したものである. 底汎化法は Muggleton による逆伴意法 [Muggleton 95] をより厳密にしかも扱いやすくしたものである. 底汎化法は、1 個の具体例を背景知識に基づいて一般化する手法であるが、複数の具体例を同時に説明する仮説を見つけるような ILP システムを構築するには、底汎化法を複数の具体例へと拡張する必要がある.

以下 a, b は定数記号, f, g は 1 引き数の関数記号, p, q, r, s, t は 1 引数の述語記号, X, Y は変数とする. 帰納推論システムに、背景知識として次の確定節プログラム B が与えられていると仮定する.

$$B = \left\{ \begin{array}{l} s(a) \leftarrow \\ s(X) \leftarrow t(X) \\ p(f(X)) \leftarrow r(X) \\ q(g(X)) \leftarrow r(X) \end{array} \right\}$$

最初に単純な場合として、正の具体例として次の単位節 E が与えられたとする.

$$E = p(f(a)) \leftarrow$$

いま、 E を説明するような仮説 H を求めることを考える. ゴール $\leftarrow p(f(a))$ の B における SOLDR 導出の帰結として $\leftarrow p(f(a))$ や $\leftarrow r(a)$ がある. すなわち、事実 $p(f(a))$ または $r(a)$ が成り立つならばゴール $\leftarrow p(f(a))$ からの SLD 反駁は成功する. また、 B において成り立つ事実の集合は、 $T_B \uparrow \omega = \{s(a)\}$ である. E の B に関する底節とは、帰結のひとつと成り立つ事実のいくつかの連言を否定した確定節

$$C_1 = p(f(a)) \leftarrow s(a),$$

$$C_2 = r(a) \leftarrow s(a)$$

をとなる.

底節 C_1, C_2 のどちらかを包摂に基づいて汎化して得られるすべての仮説 H は $B \wedge H \vdash E$ を満たす. $T_P \uparrow \omega$ を求める手法はボトムアップ評価, Semi-naive Evaluation 法 [Ullman 88] など論理プログラミングの研究で開発されている手法を用いる.

次に、正の具体例として単位節でない節、たとえば

$$F = q(g(b)) \leftarrow t(b)$$

が与えられた場合を求めることを考える. 例 F を B のもとで説明するためには、前提条件 $t(b)$ を考慮した背景知識 $B \wedge (t(b) \leftarrow)$ のもとで単位節 $q(g(b)) \leftarrow$ を説明すればよい. ゴール $\leftarrow q(g(b))$ の帰結は $\leftarrow q(g(b))$ と $\leftarrow r(b)$ であり、 $T \uparrow_{B \wedge (t(b) \leftarrow)} = \{s(a), t(b), s(b) \leftarrow\}$ である. F の底節は

$$D_1 = q(g(b)) \leftarrow s(a), t(b), s(b),$$

$$D_2 = r(b) \leftarrow s(a), t(b), s(b)$$

であり E の場合と同様に D_1, D_2 のどちらかを包摂に基づいて汎化して得られるすべての仮説 H は $B \wedge H \vdash F$ を満たす.

以上のように、与えられた一つの具体例の底節を求め、底節を一般化することにより、具体例を背景知識のもとで説明する仮説 H を求める手法が底汎化法 [Yamamoto 98a] である. ここでは、変数が現れない具体例 E, F を扱ったが、与えられる具体例に変数 Y が含まれる場合も、 Y をスコーム定数 c_Y に置き換えることによって、変数が含まれない場合と同様に扱うことができる [Yamamoto 98c].

いま、背景知識 B のもとで、 E と F を同時に説明する確定節 H をを見つけることを考える. E と F の底節の包摂に基づく最小共通汎化 [Plotkin 70] を考えると、底節の選び方によって、次の 4 つのホーン節が求まる.

$$H_1 = lgg(C_1, D_1) = \leftarrow s(a), s(X),$$

$$H_2 = lgg(C_1, D_2) = \leftarrow s(a), s(X),$$

$$H_3 = lgg(C_2, D_1) = \leftarrow s(a), s(X),$$

$$H_4 = lgg(C_2, D_2) = r(X) \leftarrow s(a), s(X).$$

ここで、 $lgg(C_i, D_j)$ は C_i と D_j の包摂に基づく最小共通汎化を表すものとする. 節 H_1, H_2, H_3 はゴール節であるので捨て、確定節である H_4 について考える. もちろん、この条件を満たす節は複数個存在する可能性があるが、興味深い仮説 [Inoue 92a] を発見する問題は記述長

などのヒューリスティクス [Rissanen 78] を与えて解決すべき別の問題であるので本稿ではふれない。

H_4 は C_2 と D_2 を共に包摂するので、 $B \wedge H_4 \vdash E$ かつ $B \wedge H_4 \vdash F$ となる。実際、 H_4 または H_4 を適当に包摂に基づき一般化した節 H 、例えば $r(X) \leftarrow s(X)$ をシステムに追加すると、

$$B \wedge H \vdash E \wedge F$$

となる。この様にして例 E と例 F を共に B のもとで説明する確定節 H を求める手法を底節交差汎化法と呼び、以降で詳細を述べる。

5.3 仮説空間の構造化

複数の正の例から構成的に仮説生成するために仮説の集合に半順序構造を導入する。半順序構造としては包摂に基づく汎化関係と背景知識を用いた相対包摂に基づく汎化関係を用いる。

5.3.1 包摂に基づく汎化

定義 [Plotkin 70] 節 C が節 D を包摂するとは、ある代入 θ が存在し、 $C\theta$ 中のリテラルがすべて $D\theta$ 中に出現することをいい、 $C \subseteq D$ と書く。

節 D から $C \subseteq D$ となる節 C を求めることを D を包摂に基づいて汎化するという。また、 C は D の包摂に基づく汎化であるという。

5.3.2 相対包摂に基づく汎化

背景知識 B に対して $B \wedge H$ という形の節形式文だけを探索する場合には、仮説空間にさらに詳細な構造を定義することができる。

Plotkin は節間の包摂関係を背景知識によって相対化している。

定義 [Plotkin 71] 節 C が節 D をプログラム B に関して相対包摂するとは、 $B \vdash \forall (D \leftrightarrow F)$ なる節 F で C が F を包摂するものが存在することをいい、 $C \subseteq D(B)$ と書く。ここで、 $\leftrightarrow$ は2つの式が論理的に同値であることを表し、 $\forall (D \leftrightarrow F)$ における節 D と F の全称束縛は取り除くものとする。

直感的には、 $C \subseteq D(B)$ は $B \wedge C \vdash D$ かつその証明において C は1回だけ使われることを意味する。

節 D と背景知識 B から $C \subseteq D(B)$ となる節 C を求めることを D を B に関する相対包摂に基づいて汎化するという。また、 C は D の B に関する相対包摂に基づく汎化であるという。

相対包摂に関して次の命題が成り立つ。

命題 5.1 確定節 C, D と確定節プログラム B に対して、 $C \subseteq D(B)$ ならば $B \wedge C \vdash D$ である。

5.4 底集合と底節

任意の具体例 E に対して、背景知識 B の底集合 $\text{BOT}(E, B)$ と底節の集合 $\text{BOTC}(E, B)$ を次のように定義する。

定義 確定節 E と確定節プログラム B に対して $\text{BOT}(E, B)$ を次のように定義する。

$$\text{BOT}(E, B) = \{L \mid L \text{ は基礎リテラルかつ } B \wedge \neg E\sigma \models \neg L\}.$$

定義 底集合 $\text{BOT}(E, B)$ 中のある正のリテラル A_0 とすべての負リテラル $\neg A_1, \neg A_2, \dots$ の選言すなわち、 $A_0 \leftarrow A_1, A_2, \dots$ を E の B に関する底節といい、 E の B に関する底節の集合を $\text{BOTC}(E, B)$ と書く。

底節と相対包摂に関して、次の定理が示されている。

定理 5.1 [Yamamoto 97a] 確定節プログラム B と確定節 E に対して、 $B \not\models E$ であるならば以下の二つの命題は同値である。

1. 仮説 H が E を B に関して相対包摂する。
2. 仮説 H はスコールム定数を含まず、かつ、 E の B に関する底節を包摂する。

命題 5.1 と定理 5.1 より、具体例 E 背景知識 B に関する底節の一つを適当に一般化することによって E を説明する仮説 H を得る。この手法を底汎化法という。

定義 底汎化法 [Yamamoto 97a]: 帰納推論システムが確定節プログラム B を背景知識として持ち、確定節 E がシステムに正例として与えられ、しかも、 $B \not\models E$ であったとする。底節の集合 $\text{BOTC}(E, B)$ 中のある節 C を包摂に基づいて汎化した節 H を出力する。ただし、節 H にはスコールム定数が含まれないものとする。

定理 5.1 より、底汎化法は、相対包摂に関して健全かつ完全である。

5.5 底節の生成手法

本節では、具体例 E の背景知識 B に関する底節を求める具体的手法を述べる。

底集合 $\text{BOT}(E, B)$ のうちの正のリテラルの集合を、 $\text{BOT}^+(E, B)$ 、負のリテラルの集合を $\text{BOT}^-(E, B)$ で表す。 $\text{BOT}^-(E, B)$ と $\text{BOT}^+(E, B)$ を求めることによって底節の集合が得ることができる。

帰納推論システムの持つ背景知識 B が確定節プログラム、システムに与えられる例 E が確定節であるとき、 $B \wedge \neg(E\sigma^-)\sigma \wedge \neg(E\sigma^+)$ は $P \wedge G$ (P は確定節プログラム、 E はゴール節) という形をしている。 B で E を説明できなければ $P \wedge G$ は充足可能である。このとき、基礎原子論理式 A を B に加えて E が説明できるようにする、すなわち $P \wedge (A \leftarrow) \wedge G$ を充足不可能にするには、 $\text{BOT}^+(E, B)$ の要素を探索すればよい。

定理 4.1 から次の系が得られる。

系 5.1 確定節プログラム B と $B \not\models E$ である確定節 E に対して、

$$\text{BOT}^+(E, B) = \{A \mid \leftarrow A \in \text{SOLDR}(P, G)\}.$$

ここで、 $P = B \wedge \neg(E\sigma^-)$ 、 $G = \neg(E\sigma^+)$ とする。

前章で与えた SOLDR 導出によって、集合 $\text{BOT}^+(E, B)$ を計算することができる。

一方、 $\text{BOT}^-(E, B)$ に関して次が成り立つ。

命題 5.2 確定節プログラム B と $B \not\models E$ である確定節 E に対して、

$$\text{BOT}^-(E, B) = \{\neg A \mid A \in T_P \uparrow \omega\}.$$

ここで、 $P = B \wedge \neg(E\sigma^-)$ とする。

$T_P \uparrow \omega$ を求める手続きは演繹推論であることから、 $\text{BOT}^+(E, B)$ はアブダクションを用いて求め、 $\text{BOT}^-(E, B)$ は演繹推論を用いて求めることができる。

5.6 包摂に基づく共通汎化

定義 確定節 C, D に対して、 $H \supseteq C$ かつ $H \supseteq D$ となる節 H を C と D の包摂に基づく共通汎化という。

特に、 C と D の包摂に基づく一般化の中では、半順序関係 $\supseteq$ のもとで最小となる節 H が必ず存在する [Plotkin 70]。そのような H を C と D の包摂に基づく最小共通汎化とい

い、 $\text{lgg}(C, D)$ と書く。すなわち、 $H \supseteq \text{lgg}(C, D)$ ならば、かつその時に限り $H \supseteq C$ かつ $H \supseteq D$ である。

確定節の包摂に基づく最小共通汎化において、次の命題が成り立つ。

命題 5.3 確定節 C と D に対して、次の二つの命題は同値である。

1. C と D の頭部の述語記号が一致する。
2. $\text{lgg}(C, D)$ が確定節である。

最小汎化の計算方法を例を用いて簡単に説明する。直感的な捉え方をすると、原子式 $A_1, \dots, A_n$ の最小汎化とは、 $A_1, \dots, A_n$ の異なる部分を変数化して得られる。原子式 $A_1, \dots, A_n$ の最小汎化のことを原子式の $A_1, \dots, A_n$ の最小汎化のことを原子式の集合 $\{A_1, \dots, A_n\}$ の最小汎化と呼ぶこともある。

例えば $\text{rev}([a, b], [b, a])$ と $\text{rev}([c], [c])$ の最小汎化を考える。 $[a, b] = [a|[b|[]]]$ 、 $[c] = [c|[]]$ であることから、 $\text{rev}([X|Y], [Z|W])$ となる。同様に、 $\text{append}([b], [a], [b, a])$ と $\text{append}([], [c], [c])$ の最小汎化は $\text{append}(X, [Y], [Z|W])$ となる。異なる部分の変数化は、できるだけ共通な変数を用いて行われる。つまり、 $\text{rev}([b], [b])$ と $\text{rev}([], [])$ の最小汎化は $\text{rev}(X, X)$ であり、 $\text{rev}(X, Y)$ ではない。

リテラルおよびリテラルの列の最小汎化も、原子式の場合と同様である。リテラルの最小汎化は、符合が異なる場合や述語が違う場合には定義されない。

たとえば、二つのリテラルの列

$$(\text{rev}([a, b], [b, a]), \text{rev}([b], [b])),$$

$$(\text{rev}([c], [c]), \text{rev}([], [])),$$

の最小汎化は、

$$(\text{rev}([A|X], [Y|Z]), \text{rev}(X, X)),$$

である。変数化にはできるだけ共通な変数が用いられるので、 X, Y, Z がそれぞれ複数の場所に現れている。

二つの節 C と D との包摂に基づく最小汎化は以下のように計算される。まず、各 C のリテラルが n 個であり、各 D のリテラルが m 個であると仮定する。つまり、

$$C = L_1 \vee \dots \vee L_n$$

$$D = L'_1 \vee \dots \vee L'_m$$

である。ここで、つぎのような二つのリテラルの列をつくる。

$$R_C = (L_1, \dots, L_1, \dots, L_n, \dots, L_n)$$

$$R_D = (L_1, \dots, L'_n, \dots, L'_1, \dots, L'_n)$$

ここで、 R_C は m 個の L_1 , m 個の $L_2, \dots, m$ 個の L_n , のように並んだ列であり、 R_D は $L'_1, L'_2, \dots, L'_m$ が m 回繰り返して並んだ列である。 R_C と R_D の列の長さはともに $m \times n$ であることに注意する。

次にこれらの二つの列の最小汎化を計算する。その結果に含まれているリテラルを集めると、始めに与えられた2つの節の最小汎化がもとまる。

$$C = gf(namihei, sazae) \leftarrow father(namihei, sazae), mother(sazae, tara), boy(tara)$$

$$D = gf(tomozo, sumire) \leftarrow father(tomozo, sumire), mother(hiroshi, maruko)$$

を考える。 C と D はそれぞれ

$$gf(namihei, sazae) \vee \neg father(namihei, sazae) \vee \neg mother(sazae, tara) \vee \neg boy(tara)$$

$$gf(tomozo, maruko) \vee \neg father(tomozo, sumire) \vee \neg mother(sumire, maruko)$$

であるので、

$$\begin{aligned} R_C = & (gf(namihei, sazae), gf(namihei, sazae), gf(namihei, sazae), \\ & \neg father(namihei, sazae), \neg father(namihei, sazae), \neg father(namihei, sazae), \\ & \neg mother(sazae, tara), \neg mother(sazae, tara), \neg mother(sazae, tara), \\ & \neg boy(tara), \neg boy(tara), \neg boy(tara)) \end{aligned}$$

$$\begin{aligned} R_D = & (gf(tomozo, sumire), \neg father(tomozo, sumire), \neg mother(sumire, maruko), \\ & gf(tomozo, sumire), \neg father(tomozo, sumire), \neg mother(sumire, maruko), \\ & gf(tomozo, sumire), \neg father(tomozo, sumire), \neg mother(sumire, maruko), \\ & gf(tomozo, sumire), \neg father(tomozo, sumire), \neg mother(sumire, maruko)) \end{aligned}$$

である。 R_C と R_D の最小汎化は、

$$\begin{aligned} lgg(R_C, R_D) = & (gf(X, Y), \text{なし}, \text{なし}, \text{なし}, \neg father(X, Z), \text{なし}, \\ & \text{なし}, \text{なし}, \neg mother(Z, Y), \text{なし}, \text{なし}, \text{なし}) \end{aligned}$$

となる。この最小汎化に含まれているリテラルを集めると、始めに与えられた2つの節の最小汎化

$$lgg(C, D) = gf(X, Y) \leftarrow father(X, Z), mother(Z, Y)$$

が得られる。

リテラルの個数が m と n の節の最小汎化には、リテラルが最大で $m \times n$ 個含まれる。3 個以上の節の最小汎化も同様に計算できる。リテラルの個数がそれぞれ n_i 個である節 k 個の最小汎化では、リテラルの個数は最大で $\prod_{i=1}^k n_i$ 個になる。

5.7 底節交差汎化法とその性質

本節では底節交差汎化法を提案し、その性質と利点を議論する。

具体例 E, F は共通の変数を持たない確定節とし、背景知識 B は確定節プログラムとする。

定義 底節交差汎化法 (Bottom Reduction Method): 具体例 E, F と背景知識 B に対して、 $B \models E$ かつ $B \models F$ とする。頭部の述語記号が等しい節 C, D を $BOTC(E, B)$ と $BOTC(F, B)$ からそれぞれ選択し、 $lgg(C, D)$ を包摂に基づいて汎化して得た仮説 H を出力する。

なお、命題 5.3 より、確定節 C と D の頭部の述語記号が異なることと、 $lgg(C, D)$ が確定節にならないことが同値であることに注意されたい。

以下で、底節交差汎化法を特徴付ける。確定節プログラム B に対して、 $H \models C(B)$ かつ $H \models D(B)$ となる確定節 H を C と D の B に関する相対包摂に基づく共通汎化という。 B に関する相対包摂に基づく最小共通汎化の一つを求めるは決定的な手続きはこれまで提案されてこなかった。

定理 5.2 背景知識 B を確定節プログラム、具体例 E, F を $B \models E$ かつ $B \models F$ である確定節とする。 $C \in BOTC(E, B)$, $D \in BOTC(F, B)$ とすると、 H が $H \models lgg(C, D)$ を満たす確定節であるならば、 H は C と D の B に関する相対包摂に基づく共通汎化である。

証明 C, D には共通のスコーム定数が出現しないことに注意する。 $H \models lgg(C, D)$ であるので、 $H \models C$ かつ $H \models D$ であり、 H にはスコーム定数は含まれない。 $C \in BOTC(E, B)$ より、 C は E の B に関する底節であり、 H は C を包摂するので定理 5.1 より、 $H \models E(B)$ である。同様にして、 $H \models F(B)$ である。□

定理 5.3 背景知識 B を確定節プログラム, 具体例 E, F を $B \models E$ かつ $B \models F$ である確定節とする. E, F の B に関する相対包摂に基づく共通汎化となる確定節 H に対して, E, F のある底節 C, D が存在して $H \supseteq \text{lgg}(C, D)$ である.

証明 $H \supseteq E(B)$ であるので, 定理 5.1 より, H は E の B に関するある底節 C を包摂する. 同様に, $H \supseteq F(B)$ より, H は F の B に関するある底節 D を包摂する. このような H は $\text{lgg}(C, D)$ を包摂するので, 定理は成り立つ. $\square$

定理 5.2 と定理 5.3 は, それぞれ底節交差汎化法の相対包摂に基づく共通汎化に対する健全性と完全性を示している. 従来の主要な帰納論理プログラミング手法も相対包摂に基づく共通汎化を求めるものであったが, 理論的な整備の遅れから, 扱うことができる規則に大きな制限があった. 定理 5.3 より, 底節交差汎化法を用いれば, このような制限を用いなくても良い.

5.8 底節交差汎化法の利点

底節交差汎化法の一つの利点として, 例として与えられる確定節 E, F の頭部の述語記号に制限を与えないことが挙げられる. 他の ILP 手法では, 例の頭部の述語記号を (明言していないが) 一つと仮定している. この様な学習は ILP の枠組みに含まれるものの, これまでは陽に扱われてこなかった.

以下に, 頭部の述語記号が異なる確定節を具体例として用いることにより, 仮説の探索空間を減少させることができる例を示す. たとえば, システムに背景知識として, 次の確定節プログラム B が与えられているとする.

$$B = \left\{ \begin{array}{l} \text{mortal}(X) \leftarrow \text{birds}(X) \\ \text{mortal}(X) \leftarrow \text{fishes}(X) \\ \text{mortal}(X) \leftarrow \text{mammals}(X) \\ \text{has_navel}(X) \leftarrow \text{mammals}(X) \end{array} \right\}$$

いま, 学習システムに二つの例

$$E = \text{mortal}(\text{socrates}) \leftarrow \text{human}(\text{socrates}),$$

$$F = \text{mortal}(\text{platon}) \leftarrow \text{human}(\text{platon})$$

が与えられたとする. ここで, $\text{BOTC}(E, B)$ の要素は,

$$C_1 = \text{birds}(\text{socrates}) \leftarrow \text{human}(\text{socrates}),$$

$$C_2 = \text{fishes}(\text{socrates}) \leftarrow \text{human}(\text{socrates}),$$

$$C_3 = \text{mammals}(\text{socrates}) \leftarrow \text{human}(\text{socrates}),$$

$$C_4 = \text{mortal}(\text{socrates}) \leftarrow \text{human}(\text{socrates})$$

であり, $\text{BOTC}(F, B)$ の要素は

$$D_1 = \text{birds}(\text{platon}) \leftarrow \text{human}(\text{platon}),$$

$$D_2 = \text{fishes}(\text{platon}) \leftarrow \text{human}(\text{platon}),$$

$$D_3 = \text{mammals}(\text{platon}) \leftarrow \text{human}(\text{platon}),$$

$$D_4 = \text{mortal}(\text{platon}) \leftarrow \text{human}(\text{platon})$$

である. E, F に対して, 底節交差汎化法によって求まる仮説は

$$H_1 = \text{birds}(X) \leftarrow \text{human}(X),$$

$$H_2 = \text{fishes}(X) \leftarrow \text{human}(X),$$

$$H_3 = \text{mammals}(X) \leftarrow \text{human}(X),$$

$$H_4 = \text{mortal}(X) \leftarrow \text{human}(X)$$

であり, 仮説の候補の数は E の底節の数と等しく, 仮説の絞り込みが不十分である.

しかし, 例として F の代わりに頭部の述語が E の述語記号と異なる節

$$F' = \text{has_navel}(\text{aristoteles}) \leftarrow \text{human}(\text{aristoteles})$$

が与えられたとする. $\text{BOTC}(F', B)$ の要素は

$$D'_1 = \text{has_navel}(\text{aristoteles}) \leftarrow \text{human}(\text{aristoteles}),$$

$$D'_2 = \text{mammals}(\text{aristoteles}) \leftarrow \text{human}(\text{aristoteles})$$

となる. このとき E と F' に対して, 底節交差汎化法を用いて求まる仮説の候補は

$$H_5 = \text{mammals}(X) \leftarrow \text{human}(X)$$

であり, E と F を用いた場合に比べて仮説を絞り込むことができる.

5.9 制限したクラスに対する底節交差汎化法

底節交差汎化法を計算機上で実行する場合、底節の数が無限になる可能性や、底節がいくらかでも長い本体を持つ可能性がある。このため、仮説の探索が有限時間で終了しないことがある。ここで、仮説の探索が終了した場合には必ず正しい答えを出力することに注意しておく。本節では仮説の探索が必ず有限時間で終了するような特別な場合を与える。

背景知識と仮説のクラスを4.5節で定義した弱縮小節からなるプログラムに制限すると、仮説の探索が有限時間で終了する。

背景知識を B 、具体例を E とし、 $P = B \wedge \neg(E\sigma^-)$ 、 $G = \neg(E\sigma^+)$ とする。集合 S の要素数を $|S|$ と表記すると、 B に関する E の底節の数は $|\text{BOT}^+(E, B)|$ に等しく、さらに、系5.1より $|\text{BOT}^+(E, B)| = |\text{SOLDR}(P, G)|$ である。 B を弱縮小節からなるプログラムに限定すると、 (P, G) からの SOLDR 導出の帰結は有限個であり、それらはすべて変数を含まない。よって、 $\text{SOLDR}(P, G)$ は有限集合になり、底節の数は有限個となる。

一方、 B に関する具体例 E の任意の底節 C の本体の長さは、 $|\text{BOT}^-(E, B)|$ に等しい。命題5.2より、 $\text{BOT}^-(E, B)$ は $T_P \uparrow \omega$ の中のすべての原子式 A を $\neg A$ に置き換えて得られる集合である。

背景知識と仮説を弱縮小節と仮定した場合 $T_P \uparrow \omega$ の代わりに原子式の集合

$$LM(P, E\sigma^+) = \{A \mid A \in T_P \uparrow \omega \text{ かつ } \|A\| \leq \|E\sigma^+\|\}$$

を用いていても、次の補題からシステムの健全性および完全性は失われない。

補題 5.1 基礎確定節 $C = A_0 \leftarrow A_1, \dots, A_n$ に対して、弱縮小節 H は $H \succeq C$ を満たしているとする。このとき、 $\|A_0\| \geq \|A_i\|$ を満たす i を $i_1, \dots, i_k$ とすれば、 $H \succeq A_0 \leftarrow A_{i_1}, \dots, A_{i_k}$ である。

証明 $H \succeq C$ より、適当な θ に対して、 $H\theta = B_0 \leftarrow B_1, \dots, B_l$ とすると、任意の $0 \leq i \leq l$ について $B_i = A_j$ なる j が存在する。 H は弱縮小節であるので、任意の $1 \leq i \leq l$ について $\|B_0\| \geq \|B_i\|$ である。□

$LM(P, E\sigma^+)$ を求める手法は論理プログラミングで開発されている手法 [Yamamoto 89] が利用できる。いま、関数記号と定数記号を P と $E\sigma^+$ に出現するものに限定することによりこの集合は有限集合となる。以上のような制限により、底節の数と底節の本体の数は有限であり、仮説の生成プロセスの停止性が保証される。図5.1に底節交差汎化法のアルゴリズムを示す。

Input: 確定節 E, F .

Given: 確定節プログラム B .

Output: E と F をともに B に関して相対包摂する確定節 H .

Procedure Bottom Recuction:

```

1   $E$  の  $B$  に関する底節  $C_1, C_2, \dots, C_n$  を求める;
2   $F$  の  $B$  に関する底節  $D_1, D_2, \dots, D_m$  を求める;
3  for  $1 \leq i \leq n$  do
4      for  $1 \leq j \leq m$  do
5          if  $C_i$  と  $D_j$  の頭部の述語記号が等しい then
6               $H := \text{lbg}(C_i, D_j)$ ;
7               $H$  の本体から弱縮小節の条件を満たさない原子式を削除する;
8               $H$  を出力する;
9          endif
10     endfor
11 endfor
end .

```

図 5.1: 底節が有限個である場合の底節交差汎化法アルゴリズム

5.10 ILP システム・BORDA

本節では、底節交差汎化法を計算機上に実装する。その実装を BORDA と呼ぶ。4.6 節で述べたように底節を構成する際に用いる SOLD R 導出が Prolog インタープリタを拡張することで、簡単に実現できることから、BORDA の実装言語として Prolog を用いた。

BORDA の Prolog を用いた実装の主要部を図 5.2 に示す。Prolog の詳細はマニュアル [SICStus 96] やテキスト等 [Clocksin&Mellish 94, Sterling&Shapiro 86] を参照されたい。

述語 `soldr/3` については前章で述べた通りである。

手続き `botc(E,B,C)` は例 E の背景知識 B に関する底節 C を、前節で述べた近似方法を用いて計算する。述語 `botc/3` で用いる中間述語 `skolemize/2` は、節中の変数をスコールム定数に置き換える述語である。手続き `add(G,B,P)` により、ゴール G 中に現れるすべてのリテラルを背景知識 B に追加したプログラム P を得る。手続き `lm(P,A,M)` はプログラム P と原子式 A から、原子式の集合 $M = LM(P,A)$ を計算する。

手続き `bottomReduction(E,F,B,H)` は、背景知識 B に関して、具体例 E と F をともに相対包摂する節 H を計算する。手続き `lgg(C,D,H)` は節 C と節 D の包摂に基づく最小共通汎化 H を計算する。 E と F の底節から頭部の述語記号が同じ C と D を見つける探索には、述語 `same_head_pred(C,D)` と Prolog のバックトラック機構を用いている。

次に、BORDA の実行例を示す。

```
| ?-E=[p(f(a)),:-],
    F=[q(g(Y)),:-,t(Y)],
    B = [ [s(a),:-],
          [s(X),:-,t(X)],
          [p(f(X)),:-,r(X)],
          [q(g(X)),:-,r(X)] ],
    bottomReduction(E,F,B,H).
```

```
H = [r(_A),:-,s(a),s(_A)] ?
yes
```

これは、仮説 $H = r(A) \leftarrow s(a), s(A)$ が、具体例 $E = p(f(a)) \leftarrow$ と $F = q(g(Y)) \leftarrow t(Y)$

```
bottomReduction(E,F,B,H):-
    botc(E,B,C), botc(F,B,D),
    same_head_pred(C,D), lgg(C,D,H).

botc(E,B,[CHead,:-|Cbody]):-
    skolemize(E,[EHead,:-|EBody]), add(EBody,B,P),
    lm(P,EHead,CBody),!, soldr([EHead],P,[CHead]).

soldr([],_Prog,[]).
soldr([G|Gs],Prog,L):-
    copy_term(Prog,Prog2), member([G,:-|Bs],Prog2),
    append(Bs,Gs,NewG), soldr(NewG,Prog,L).
soldr([G|Gs],Prog,[G]):-
    soldr(Gs,Prog,[]).
soldr([L|Gs],Prog,[L]):-
    soldr(Gs,Prog,[L]).
```

図 5.2: 底節交差汎化法を用いて仮説を求める Prolog プログラム

と 5.2 節で用いた背景知識 B

$$B = \left\{ \begin{array}{l} s(a) \leftarrow \\ s(X) \leftarrow t(X) \\ p(f(X)) \leftarrow r(X) \\ q(g(X)) \leftarrow r(X) \end{array} \right\}$$

から得られたことを示している。

5 章のまとめ

本章では確定節プログラムに対する新しい ILP 手法である底節交差汎化法を提案し、同手法を BORDA とよばれる ILP システムとして実装した。

底節交差汎化法は、背景知識となる論理プログラムと具体例となる複数の確定節から、具体例を説明する仮説を計算する手法であり、同手法は、SOLDR 導出・ボトムアップ評価・包摂に基づく共通汎化という三つの手法に基づいている。SOLDR 導出は、論理プログラムを用いた発想推論(アブダクション)の実現手法である。ボトムアップ評価[Ullman 88]は演繹データベースの分野で開発された演繹推論(デダクション)の手法である。包摂に基づく共通汎化[Plotkin 70]は帰納論理プログラミングの研究分野で開発された一般化の手法である。これら三つの手法に基づいて仮説を生成することで、同手法による仮説生成の過程を論理的に意味付けた。

また、底節交差汎化法で求まる仮説の特徴付けを、相対包摂に基づく共通汎化[Plotkin 71]によって与えた。すなわち、同手法で求まる仮説はすべて与えられた具体例の相対包摂に基づく共通汎化である。逆に、与えられた具体例の相対包摂に基づく共通汎化は、すべて底節交差汎化法によって求めることができる。従来の主要な帰納論理プログラミング手法[Muggleton 95]も相対包摂に基づく共通汎化を求めるものであったが、理論的な整備の遅れから、扱うことができる規則には大きな制限があった。本論文では、底節交差汎化法を提案することによりこの制限を取り払った。

底節交差汎化法に関連する理論的研究としてRLGG[Buntine 88]や $CIG_{\geq \Sigma}$ [Page&Frish 92]が挙げられる。両者とも述語記号が異なる具体例を扱えない。それに対し、底節交差汎化法は述語記号が異なる具体例を扱うことができる。

底節交差汎化法において、異なる述語の具体例を用いると、一般に仮説の探索空間を減少させることを考察した。本章では与えられる具体例が二つの場合のみを示したが、三つ以上の具体例に関しても同様に扱うことができる。

第6章

帰納論理プログラミングとしての正則言語の学習

本章では、まず、正則言語とその学習に関する既知の定理について述べ、Angluinによって提案された正則言語の学習アルゴリズムを概説する。そして、正則言語の学習問題を帰納論理プログラムの観点から考察し、それを述語発見を用いた帰納論理プログラミングに発展させるための準備を行う。

6.1 正則言語とその学習に関する既知の定理

有限オートマトンとは5つ組 $M = (K, \Sigma, \delta, q_0, F)$ のことである。ここで K は空でない有限集合で、その要素は状態と呼ばれる。 Σ はアルファベットで、 δ は $K \times \Sigma$ から K への関数で遷移関数とよばれる。また、 $q_0 \in K$ は初期状態、 F は K の部分集合であり、その要素は最終状態と呼ばれる。遷移関数 δ を

$$\delta(q, \lambda) = q \quad (6.1)$$

$$\delta(q, ax) = \delta(\delta(q, a), x) \quad (6.2)$$

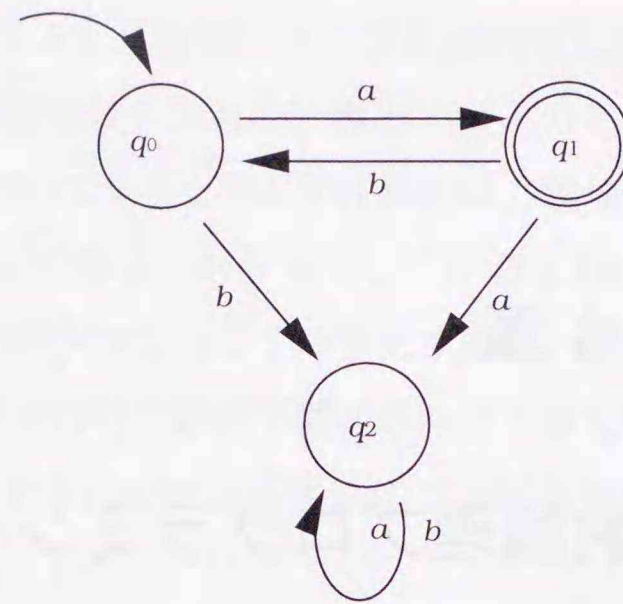
によって、 δ を $K \times \Sigma^*$ から K への関数に拡張する。ここで、 λ は長さが0である語(空語)を表す。

語 $w \in \Sigma^*$ は $\delta(q_0, w) \in F$ であるとき、有限オートマトン M によって受理されるという。また $L(M) = \{w \in \Sigma^* | \delta(q_0, w) \in F\}$ とする。

Σ^* の部分集合 R は $R = L(M)$ となる M が存在するときに限り正則であるという。

例 6.1 $K = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, $F = \{q_1\}$, 図6.1が表す δ を遷移関数とするオートマトン $M = (K, \Sigma, \delta, q_0, F)$ を考える。 M の状態遷移図を図6.2に示す。すると、 M の表す言

δ	a	b
q_0	q_1	q_2
q_1	q_2	q_0
q_2	q_2	q_2

図 6.1: M の遷移関数 δ 図 6.2: M の状態遷移図

語 $L(M)$ は

$$L(M) = \{a, aba, ababa, ababa, \dots\} = \{a(ba)^i \mid i \geq 0\}$$

である.

定義 Σ^* 上の関係 R は

$$xRy \implies \text{任意の } z \in \Sigma^* \text{ に対して } xzRyz$$

を満たすとき, 右不変 (right invariant) であるという. また, Σ^* 上の同値関係 R は, R による同値類の数が有限であるとき, 有限指数 (finite index) であるという.

次に示す Nerode の定理と呼ばれるは正則言語を特徴付けを与える.

定理 6.1 [有川&宮野 86] 次の3つの命題は同等である.

1. 集合 $L \subseteq \Sigma^*$ は正則である.
2. L はある有限指数で右不変な同値関係 R による同値類の和として表現される.
3. 関係 $\equiv_L$ は有限指数である. ここで $\equiv_L$ は「 $x \equiv_L y \iff$ 任意の $z \in \Sigma^*$ に対して, $xz \in L$ と $yz \in L$ が同等である」によって定義される関係である.

正則言語の与えられた正例と負例からの帰納推論について Gold より次の定理が与えられている.

定理 6.2 [Gold 78] 正則言語は与えられた正負の例からその言語を表現する有限オートマトンを帰納推論する問題は NP-完全である.

この定理により, 正則言語は与えられた正負の例からではその言語を表現する有限オートマトンを効率よく帰納推論することはできないことがわかる.

一方, Angluin によって正則言語は質問を用いることにより, 効率よくできることが示された.

定理 6.3 [Angluin 87] 正則言語は所属性質問と等価性質問に対して答える教師を用いると, その言語を表現する有限オートマトンをパラメータ m と n の多項式時間で必ず学習することができる. ここで, m はその言語を表現する有限オートマトンの状態数であり, n は反例として与えられた語の長さの最大値である.

所属性質問とは, 言語 L に語 c が入っているかどうかを質問することである. 所属性質問に対し, 教師は $c \in L$ である場合は yes を返しそうでない場合は no を返す. また, 等価性質問とは, 教師に推測のオートマトン M を提示し, M が正しいかどうかを質問することである. M が正しい場合は, 教師は yes を返し, そうでない場合は反例を返す. 反例とは, 推測のオートマトン M の表す言語に入っているが学習の対象としている (教師の持っている) 正則言語 L に入っていない語 (正の反例), および学習の対象としている正則言語 L に入っているが, 推測のオートマトン M の表す言語に入っていない語 (負の反例) のことである. これらの定理から正則言語の学習には, 学習者が積極的に質問することが学習の効率を向上させることが明らかである.

6.2 正則言語の学習アルゴリズム

本節では, 正則言語の学習アルゴリズム [Angluin 87, Kearns&Vazirani 94] について概説する. Angluin の学習アルゴリズムでは, 観測表と呼ばれる表を用いているが, ここでは, より一般的な状態分類木 (classification tree) という決定木を用いた正則言語のアルゴリズム [Kearns&Vazirani 94] について述べる.

定義 状態分類木 (classification tree) は図 6.3 のように葉が状態指定語 (state access string) で, 葉でないノードが識別語 (distinguishing string) であるような2分木である.

$\text{Shift}(s, T)$ は, 状態分類木 T と所属性質問から, 文字列 s の状態指定語を求める手続きである. 文字列 s の後ろに識別語を付け加えた文字列 sw に対して所属性質問を行い, yes が返ってくると右の子のノードに進み, no が返ってくると左の子のノードに進んで, 葉 t に至ったとき状態指定語 t を返す.

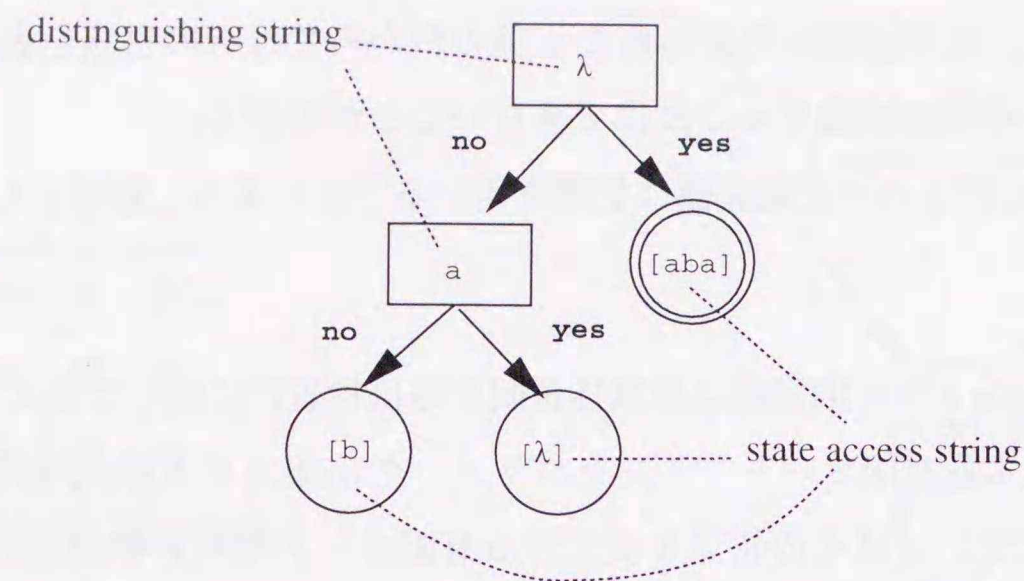


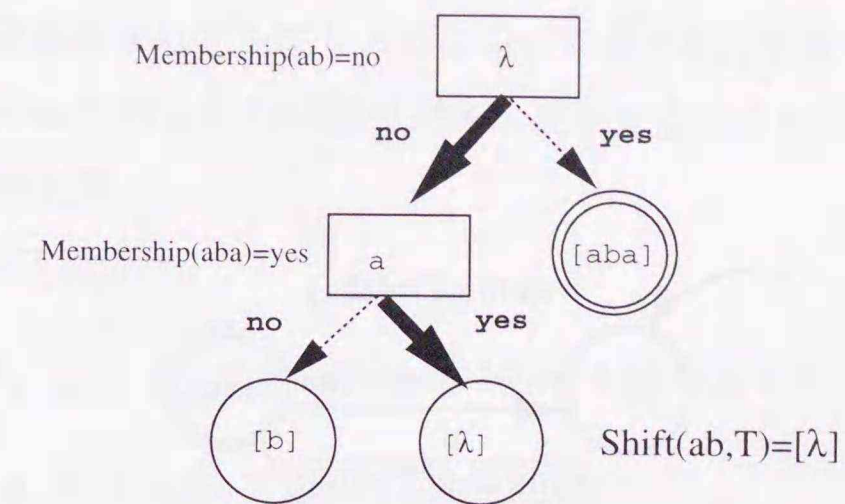
図 6.3: 状態分類木

例 6.2 図 6.3 の状態分類木 T と語 ab に対して $\text{Shift}(ab, T)$ の実行を考える. 語 w の所属性質問に対する答えは, 例 6.1 における言語 $L(M)$ に従うとする. つまり, 語 $w \in L(M)$ のとき yes が返され, そうでないとき no が返されるとする. まず T の根から実行を始める. T の根にラベル付けられている識別語は λ であるので, 語 ab の所属性質問を行う. $ab \notin L(M)$ であるので左の子のノードに進む. このノードには識別語は a がラベル付けられているので, 語 ab に a を接続した語 aba の所属性質問を行う. $aba \in L(M)$ であるので右の子のノードに進む. このノードは葉であり, 状態指定語 $[ab]$ を出力し, $\text{Shift}(ab, T)=[\lambda]$ となる. 図 6.4 にその様子を示す. 語 $abab$ についても, 同様に $\text{Shift}(abab, T)=[\lambda]$ となる.

この例において鍵となるのは, $ab \equiv_{L(M)} abab \equiv_{L(M)} \lambda$ であるので, $\text{Shift}(ab, T)=\text{Shift}(abab, T)=[\lambda]$ となっていることである. このことは状態指定語は Σ^* の $\equiv_{L(M)}$ による同値類の代表元であると考えると理解しやすい.

次に状態分類木 T から推測のオートマトン M_T を作る手続き $\text{Hypo}(T)$ について述べる. $\text{Hypo}(T)$ の手続きは以下の通りである.

- 状態の集合 K_T を状態指定語の集合とする.
- 遷移関数を $\delta_T([s], a) = \text{Shift}(sa, T)$ とする.
- 初期状態を $\text{Shift}(\lambda, T)$ とする.
- 終了状態 F_T を T のルートの右の子の下にある状態指定語の集合とする.

図 6.4: 手続き $\text{Shift}(s, T)$

例 6.3 図 6.3 の状態分類木 T に対する $\text{Hypo}(T)$ の実行を考える. 例 6.2 と同様に語 w の所属性質問に対する答えは, 例 6.1 における言語 $L(M)$ に従うとする. すると,

$$\text{状態の集合 } K_T = \{[\lambda], [b], [aba]\}$$

遷移関数

$$\delta_T([\lambda], a) = \text{Shift}(a, T) = [aba]$$

$$\delta_T([\lambda], b) = \text{Shift}(b, T) = [b]$$

$$\delta_T([b], a) = \text{Shift}(ba, T) = [b]$$

$$\delta_T([b], b) = \text{Shift}(bb, T) = [b]$$

$$\delta_T([aba], b) = \text{Shift}(abab, T) = [\lambda]$$

$$\delta_T([aba], a) = \text{Shift}(abaa, T) = [aba]$$

$$\text{初期状態} = \text{Shift}(\lambda, T) = [\lambda]$$

$$\text{終了状態 } F_T = \{[aba]\}$$

であり, $M_T = (K_T, \{a, b\}, \delta_T, [\lambda], F_T)$ となる. 図 6.5 に T から作られる推測のオートマトン $\text{Hypo}(T)$ を示す.

ここで, $\text{Hypo}(T)$ と所属性質問の対象とした例 6.1 のオートマトン M が同形であることに注意する. 同形である二つのオートマトンが受理する言語は等しい.

次に, 状態分類木を構成する手法について述べる. 状態分類木は初期状態分類木から始めて, 等価性質問に対する反例 w によって状態分類木を更新してゆくことで構成される.

初期状態分類木とは, 根が状態指定語 $[\lambda]$ でラベル付けられた葉である状態分類木をいう. 初期状態分類木 T に対するオートマトン M_T を図 6.6 に示す. 空語 (λ) への所属性質問の答が yes であるとき右のオートマトン, yes であるとき左のオートマトンを M_T とする.

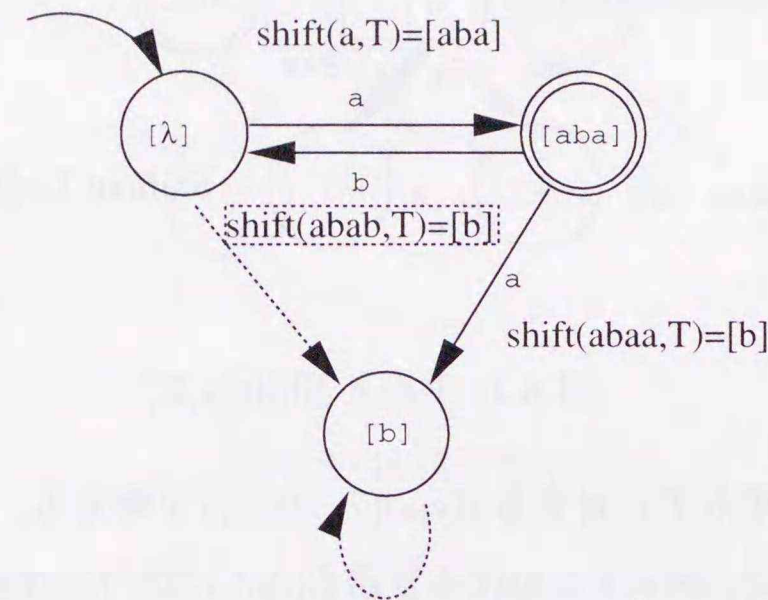
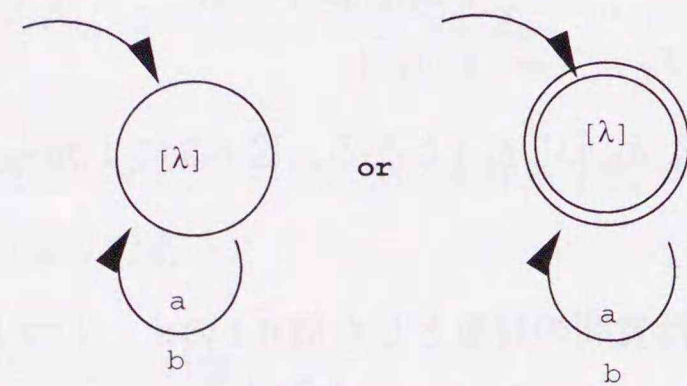
図 6.5: Hypo(T) により状態分類木から作られるオートマトン

図 6.6: 初期状態分類木から作られるオートマトン

次に、状態分類木を更新するアルゴリズムについて述べる。手続き $\text{Update}(w, T)$ を用いて、等価性質問に対する反例 w から状態分類木を更新する。この際、状態分類木のノードと葉をそれぞれ一つ増える。

手続き Update を以下に示す。

- T を状態分類木, w を M_T の等価性質問に対する反例とする;
- 語 $w = a_1 a_2 \dots a_n$ とする ($n \geq 0$);
- w の接頭語 $a_1 a_2 \dots a_i$ ($1 \leq i \leq n$) において

$$\text{Shift}(a_1 a_2 \dots a_i, T) = [s_i]$$

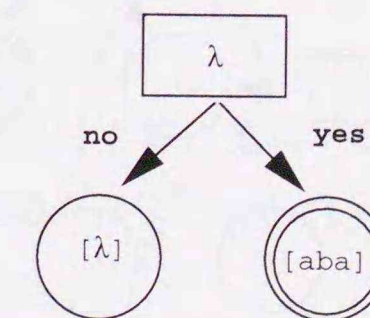
$$\delta_T([\lambda], a_1 a_2 \dots a_i) = [t_i]$$

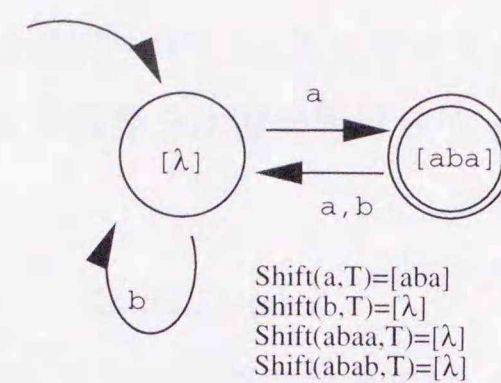
とする;

- j を $[s_j] \neq [t_j]$ となる最小の j とする;
- T において葉 $[s_j]$ と葉 $[t_j]$ の共通の親となるノードのうち最も下にあるノードにラベル付けられた識別語を d とする;
- 状態 T における葉 $[s_{j-1}]$ を識別語 $a_j d$ がラベル付けられたノードに置き換え, その子を状態指定語 $[s_{j-1}]$ と新しい状態指定語 $[a_1 a_2 \dots a_{j-1}]$ とする;

次の例で手続き $\text{Update}(w, T)$ によって、状態分類木を更新する様子を示す。

例 6.4 例 6.1 の $L(M)$ を学習の対象とし、反例を用いて図 6.7 の状態分類木 T_1 を更新することを考える。手続き Hypo によって T_1 から作られる推測のオートマトン M_{T_1} を図 6.8 に示す。いま、推測のオートマトン M_{T_1} は、学習の対象を正しく表していない。 M_{T_1} について

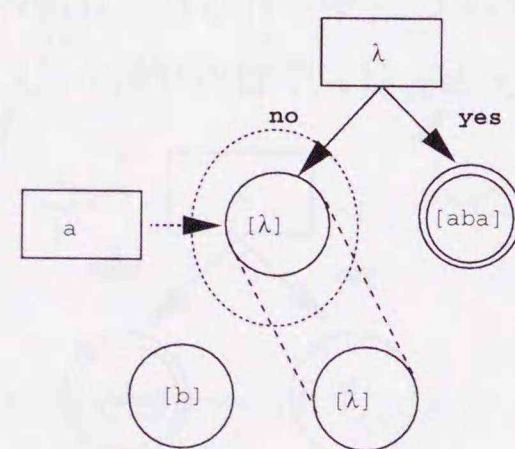
図 6.7: 状態分類木 T_1

図 6.8: $\text{Hypo}(T_1)$ により作られるオートマトン M_{T_1}

等価性質問を行うと、学習の対象と推測のオートマトン M_{T_1} が一致していないので教師は反例を返す。その反例を ba としよう。反例 ba の各接頭語 s の M_{T_1} における遷移先 $\delta_{T_1}([λ], s)$ と $\text{Shift}(ba, T_1)$ の組は、次の表が示すようになる。

i	反例の接頭語	$\text{Shift}(s, T_1)$	M_{T_1} における s の遷移先
0	λ	$[\lambda]$	$[\lambda]$
1	b	$[\lambda]$	$[\lambda]$
2	ba	$[\lambda]$	$[aba]$

この場合、 $[s_i] = \text{Shift}(ba, T_1) = [\lambda] \neq \delta_{T_1}([\lambda], ba) = [aba] = [t_i]$ であるので、手続き Update における変数 j は 2 となる。 T_1 において、葉 $[\lambda]$ と $[aba]$ の最も下にある共通の親はノード λ であるので、手続き Update における変数 d は λ となる。手続き Update は、 $a \cdot d = a$ であるので、葉 $[\lambda]$ を新しいノード a に置き換え、葉 $[\lambda]$ と新しい葉 $[b]$ を識別語 a の子とする。 $\lambda \cdot a \in L(M)$ であり $b \cdot a \notin L(M)$ であるので、ノード a の右の子は葉 $[\lambda]$ 、左の子は葉 $[b]$ となる。図 6.9 にその様子を示す。

図 6.9: 手続き $\text{Update}(w, T)$

Input: アルファベットの集合 Σ .

Given: 所属性質問と等価性質問.

Output: Σ 上のオートマトン M .

Procedure *LearningDFA*:

- 1 T を初期状態分類木とする;
 - 2 推測のオートマトンを M_T として等価性質問を行う;
 - 3 **repeat**
 - 4 $w :=$ 反例;
 - 5 手続き $\text{Update}(w, T)$ によって T を更新する;
 - 6 手続き $\text{Hypo}(T)$ により、オートマトン M_T を作る;
 - 7 M_T を提示し、等価性質問をする;
 - 8 **until** 等価性質問に対して yes;
 - 9 M_T を出力して停止する;
- end .**

図 6.10: 正則言語の学習アルゴリズム

所属性質問と等価性質問を用いた正則言語の学習アルゴリズムの概略を図 6.10 に示す。次の例で、学習アルゴリズムの実行例を示す。

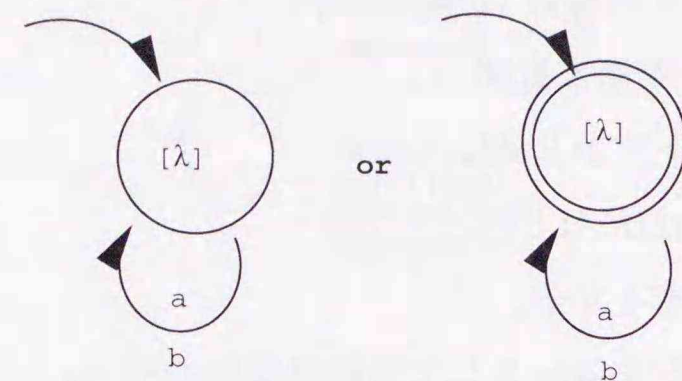
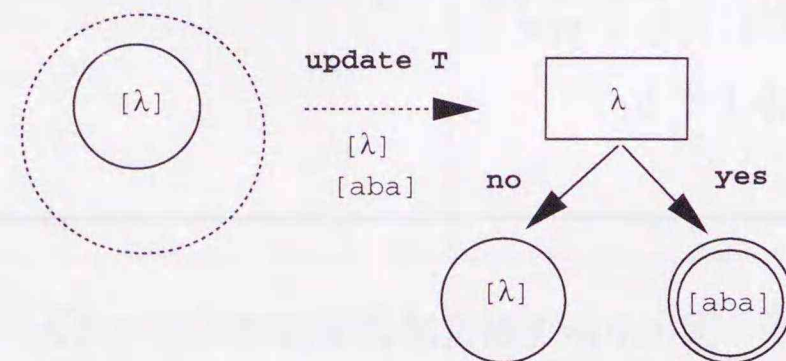
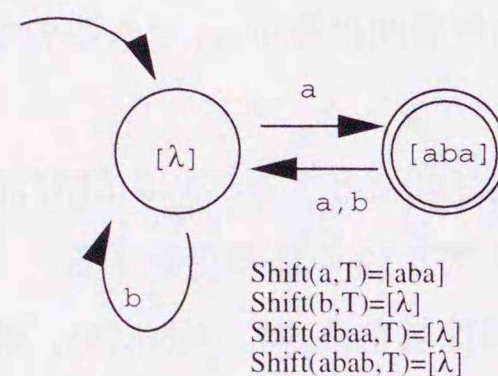
例 6.5 学習の対象となる言語を例 6.1 の $L(M)$ とする。

アルゴリズムは空語 (λ) に対して所属性質問を行い、はじめの推測のオートマトンを作る。この場合空語 (λ) への所属性質問の答が no であるので、左のオートマトンを推測として、等価性質問を行う。

反例として、文字列 aba が返されたとする。 λ は $L(M)$ の要素ではなく、 aba は $L(M)$ の要素であるので、状態分類木を図 6.12 の様に更新する。

次に状態分類木を用いて遷移関数を決定し (図 6.13)、推測のオートマトンとして等価性質問を行う。

等価性質問の答として、文字列 ba が返ってきたとする。 $\text{Update}(ba, T_1)$ によって、 T_2 を求める。すると、 T_2 は図 6.3 で示した状態分類木となる。そして、 T_2 から作られる推測のオートマトンは例 6.3 のオートマトンとなり、対象としていたオートマトンを学習した。等

図 6.11: 推測のオートマトン M_0 図 6.12: $\text{Update}(aba, T_0)$ の様子図 6.13: 推測のオートマトン M_{T_1}

価性質問に対して, yes が返ってきて学習アルゴリズムは停止する.

この例では, 9 種類の所属性質問を行い, 3 回の等価性質問を行った. 最後の等価性質問に yes が返されたので, 用いた反例は 2 個である.

以上のようにオートマトンの学習アルゴリズムは, 述語名分類木を用いて文字列を有限個の同値類に分割し, その各同値類にたいして推測のオートマトン状態を割り当てている. 述語名分類木による同値類分割は, Nerode の定理でいうところの $\equiv_L$ による同値類分割に対応する. また, 学習の過程において, 反例が返る度に新しい状態を一つずつ導入し, その状態の数が学習の対象 L を表す状態数最小オートマトン M の状態数と一致したとき, 必ず学習は成功する.

6.3 正則言語を表現する論理プログラム

本節では, 正則言語を表現する論理プログラムのクラスを与える. 正則言語を論理プログラムで表現した上で正則言語の学習を取り扱う研究は, Shapiro や Ishizaka によってされてきた [Shapiro 81, Ishizaka 89]. 本節では彼らとは違った形の表現方法を用いる.

Σ 上の正則言語の論理プログラム表現を次で与える.

定義 M を L を表す有限オートマトンとする. 論理プログラム P_M を次のように定義する.

1. 状態の集合が $\{q_1, \dots, q_n\}$ であるとき, 論理プログラムの述語の集合 P_M の述語記号の集合を $\{q'_1, \dots, q'_n\}$ とする. ただし, 各 q_i は一引数の述語記号であるとする.
2. 初期状態が q_i であるとき, P には $q'_i(0) \leftarrow$ という単位節が含まれるものとする.
3. Σ 中の文字 a に対して 1 引数の関数記号 a' 用意する.
4. 各遷移関数

$$\delta(q_i, a) = q_j$$

に対して, P には,

$$q'_j(a'(X)) \leftarrow q'_i(X)$$

という確定節が含まれるとする.

5. 終了状態の部分集合が $\{q_{i1}, q_{i2}, \dots, q_{ik}\}$ であるとき P には,

$$\begin{aligned} R(X) &\leftarrow q'_{i1}(X) \\ R(X) &\leftarrow q'_{i2}(X) \\ &\vdots \\ R(X) &\leftarrow q'_{ik}(X) \end{aligned}$$

という k 個の確定節が含まれているとする.

命題 6.1 M を言語 L を表すオートマトンとする. 項の集合 $\{t \mid R(t) \in M(P_M)\}$ において, 定数記号 0 を空語, 項 $a'(w')$ を文字列 wa に解釈すると言語 L は一致する. ここで, w は項 w' の表す文字列である.

例 6.6 例 6.1 の M の表す言語 $L(M)$ を考える. 論理プログラム P_M を

$$P_M = \left\{ \begin{array}{l} q'_0(0) \leftarrow \\ q'_1(a'(X)) \leftarrow q'_0(X) \\ q'_2(b'(X)) \leftarrow q'_0(X) \\ q'_2(a'(X)) \leftarrow q'_1(X) \\ q'_0(b'(X)) \leftarrow q'_1(X) \\ q'_2(a'(X)) \leftarrow q'_2(X) \\ q'_2(b'(X)) \leftarrow q'_2(X) \\ R(X) \leftarrow q'_1(X) \end{array} \right\}$$

とすると, P_M の最小エルブランモデル $M(P_M)$ は

$$\left\{ \begin{array}{l} R(a'(0)), R(a'(b'(a'(0)))), R(a'(b'(a'(b'(a'(0)))))), \dots, \\ q'_0(0), q'_0(b'(a'(0))), q'_0(b'(a'(b'(a'(0)))))), \dots, \\ q'_1(a'(0)), q'_1(a'(b'(a'(0)))), q'_1(a'(b'(a'(b'(a'(0)))))), \dots, \\ q'_2(b'(0)), q'_2(a'(a'(0))), q'_2(b'(b'(0))), \dots \end{array} \right\}$$

となり, $L(M)$ は P_M における述語 R で表現される.

6.4 正則言語を表す論理プログラムの学習問題

前節の議論と Gold の結果から, 論理プログラム P_M は与えられた正負の具体例からでは効率よく帰納推論できないことが明らかである.

一般に ILP は与えられた正と負の例から正の例のみを論理的に説明する規則を帰納的に計算する枠組みである. 条件 1.1 より,

$$P \vdash R(s_1) \wedge \dots \wedge R(s_n)$$

$$P \not\vdash R(t_1) \vee \dots \vee R(t_n)$$

である. ここで, $R(s_i)$ を正の例, $R(t_j)$ を負の例とする.

いま, このような枠組みでは補助述語の生成を許した場合, 効率的な学習ができないことがわかった.

一方, オートマトンを表す論理プログラムは所属性質問と等価性質問を用いると学習が可能であることは, 明らかである. ここで, オートマトンの学習アルゴリズムをより一般的な論理プログラムの学習に一般化するための議論を行う.

まず, 定義から, 文字 a, b 上の正則言語を表す論理プログラムは, 次のような形をしている.

1. P に含まれる単位節は

$$q'_i(0) \leftarrow$$

という形をしている.

2. P に含まれる確定節は

$$q'_j(a'(X)) \leftarrow q'_i(X) \text{ または } q'_j(b'(X)) \leftarrow q'_i(X)$$

の形をしているか

$$R(X) \leftarrow q'_{i1}(X)$$

の形をしている.

この形の確定節を有限個用いて文字 a, b 上のすべての正則言語を表すことができる.

次の点も注意する必要がある. いま, $\{a, b\}$ 上の正則言語 L を表すプログラム P を考える. $\{a, b\}$ 上の文字列を表す基礎項 s, t について, θ_1 を代入 $\{X/s\}$ と θ_2 を代入 $\{X/t\}$ とする. すると Nerode の定理より, 関数記号 a', b' と変数 X を用いた任意の項 α に対して,

$$L(\alpha)\theta_1 \in M(P) \text{ ならばかつそのときに限り } L(\alpha)\theta_2 \in M(P)$$

となる.

次章では以上の点に注意して、本章で与えた正則言語を表すプログラムの構文に対する制約を緩めることにより、関係を表すプログラムに一般化する。そして、その質問学習アルゴリズムを与える。

6 章のまとめ

本章では、正則言語の学習問題を帰納論理プログラムの観点から考察し、それを述語発見を用いた論理プログラムの学習へ発展させるための準備を行った。Angluinによるオートマトンの学習を帰納論理プログラムとしてとらえた場合、新しい補助述語の導入を用いた帰納推論と見なすことができる。帰納論理プログラミングにおいて、補助述語の導入問題は、困難であると言われている問題であるが、次章で正則言語の学習アルゴリズムを一般化し、正則な論理プログラムというクラスの述語発見を用いた学習アルゴリズムを与える。

第 7 章

スキーマと質問を用いた述語発見による論理プログラムの構成的学習アルゴリズム

本章では、論理プログラムのスキーマを用いた補助述語発見問題を、正則言語の学習に関する理論的な研究成果の一般化として議論し、スキーマと質問を用いて効率良く学習することができる補助述語を含む論理プログラムのクラスを与える。そのために述語名決定木とその更新アルゴリズムとして新たな形の矛盾点追跡アルゴリズムも与える。

7.1 スキーマを用いた帰納論理プログラミング

補助述語を効率よく発見するための一つのアプローチとして、あらかじめプログラムのスキーマを与える手法が提案されている [Stahl 95]。本章では CIA [Raedt & Bruynooghe 92] や RDT [Wirth & O'Roke 91] と同様に、スキーマを二階の論理における述語変数を用いた節の形で表す。例えば、 $\mathcal{P}_0, \mathcal{P}_1$ を述語変数とした節形式文

$$\Sigma = \left\{ \begin{array}{l} \mathcal{P}_0(f(X)) \leftarrow \mathcal{P}_1(X) \\ \mathcal{P}_0(g(X)) \leftarrow \mathcal{P}_1(X) \end{array} \right\}$$

をスキーマとし、 $\mathcal{P}_0, \mathcal{P}_1$ に述語記号を代入した節を対象の推論に用いるのである。しかし、プログラムのスキーマに関数記号の出現を許すと、たとえスキーマを使っても効率的な学習は不可能である。実際、上のスキーマ Σ を用いて、さらに一つの基礎単位節の使用を認めた表現言語を用いれば、ある定数記号 c を空語、 f, g をそれぞれ文字列 w に先行文字 $\bar{f}, \bar{g}$ を付加する関数と解釈することにより、 $\{\bar{f}, \bar{g}\}$ 上の文字列の正則言語全体を表すことができる。正則言語は正負の具体例からでは効率良く学習できないことが知られている [Gold 78] ので、スキーマを与えただけでは効率の良い補助述語生成は困難であることがわかる。

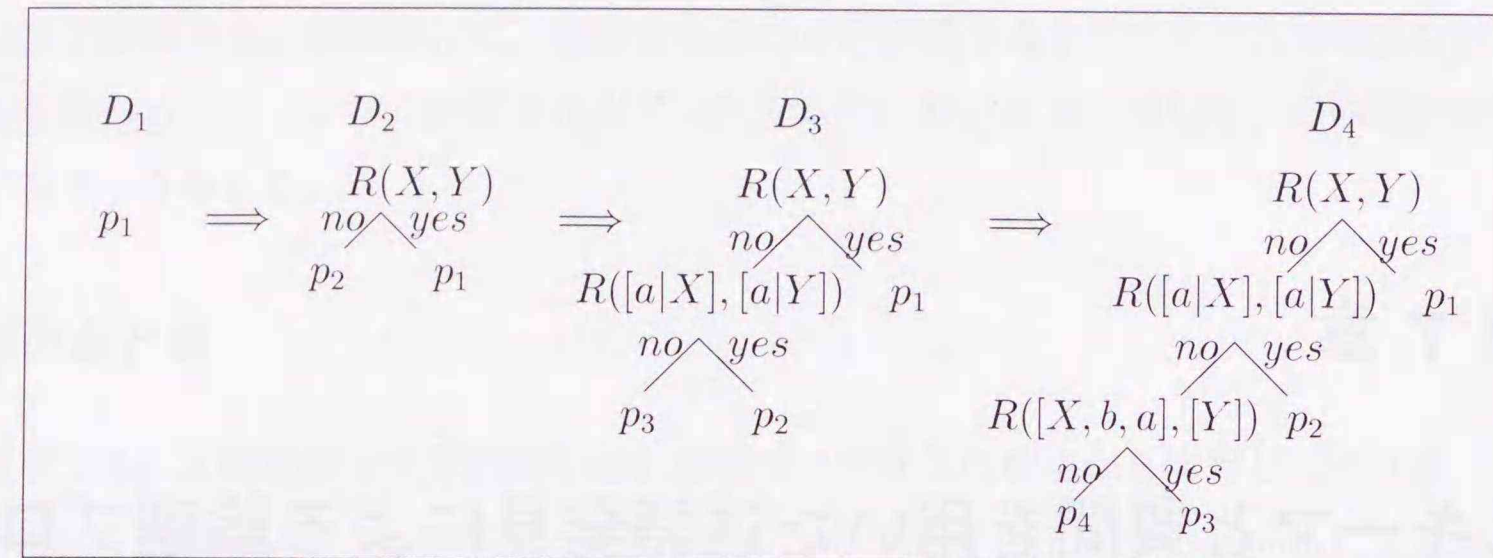


図 7.1: 述語名決定木の更新の様子

本章ではこの問題を、学習目標の関係 R を知る教師を仮定し、教師に対する質問を学習アルゴリズムに許すことにより解く。Angluin は質問を用いることで正則言語を効率良く学習するアルゴリズム [Angluin 87] を与え、さらに一般的に学習における質問の重要性を示している [Angluin 88]。Sakakibara は Angluin の仕組みを論理プログラムの学習に応用して、線形 1 引数論理プログラムの学習アルゴリズムを与えた [Sakakibara 90]。また、著者らも線形 1 引数論理プログラムを多引数に拡張した多重線形論理プログラムを効率良く学習するアルゴリズムを構成した [Ito&Yamamoto 92]。本章であたえる解は、スキーマを用いることでこれらの結果を一般化したものである。

7.2 学習アルゴリズムの戦略

学習目標の関係を文字列の組 $\langle a(ba)^{i+1}, a(ba)^i \rangle$ をリスト表現にした次の基礎項の 2 項関係 R とする。

$$R = \left\{ \begin{array}{l} \langle [a, b, a], [a] \rangle, \\ \langle [a, b, a, b, a], [a, b, a] \rangle, \\ \langle [a, b, a, b, a, b, a], [a, b, a, b, a] \rangle, \\ \vdots \end{array} \right\}$$

つまり、タプル $\langle [a, b, a], [a] \rangle$ や $\langle [a, b, a, b, a], [a, b, a] \rangle$ などが R の正の具体例であり、 $\langle a, a \rangle$, $\langle b, b \rangle$, $\langle [a|a], [b] \rangle$, $\langle [a, b], [a] \rangle$, $\langle [[a, b], a], [a] \rangle$ などが R の負の具体例である。

学習アルゴリズムにはスキーマ Σ_1 として

$$\{ \mathcal{P}_0([A|X], [B|Y]) \leftarrow \mathcal{P}_1(A, B), \mathcal{P}_2(X, Y) \}$$

が与えられ、スキーマ Σ_1 の具体化であるプログラム $P \in \mathcal{C}(\Sigma_1)$ のうち、 R を表現するものを求める。 Σ_1 の形から 2 引数の述語についてのみ考えればよい。

はじめに学習アルゴリズムは、推測として $H_0 = \phi$ を教師に提示し、 H_0 が目標の関係を表すプログラムであるかどうか等価性質問をする。 H_0 は R を表現するプログラムでないで、教師は反例、例えば

$$CE_1 = \langle [a, b, a], [a] \rangle$$

を返す。 $H_0 = \phi$ なので CE_1 は正の反例である。

まず、 CE_1 を Σ_1 を用いて部分タプルに分解し、タプルの集合

$$S_1 = \left\{ \begin{array}{l} \langle a, a \rangle, \\ \langle [b, a], [] \rangle, \\ \langle [a, b, a], [a] \rangle \end{array} \right\}$$

を作成する。このようなタプルを推測のプログラムを作るためのサンプルとして用いるので、サンプルタプルと呼ぶ。つぎに、 S_1 中のサンプルタプルに適当に述語記号を割り当て、原子式の集合 (テンポラリーモデル) をつくる。はじめはすべてのサンプルタプルにひとつの述語記号を割り当てる。今の場合、まだ述語記号は使っていないので、新たな述語記号 p_1 を用意し、テンポラリーモデル

$$I_1 = \left\{ \begin{array}{l} p_1(a, a), \\ p_1([b, a], []), \\ p_1([a, b, a], [a]) \end{array} \right\}$$

を構成する。そして、 I_1 と Σ_1 から推測プログラムとして次のようなプログラム H_1 を作る。

$$\begin{aligned} R(X, Y) &\leftarrow p_1(X, Y) \\ p_1(a, a) &\leftarrow \\ p_1([b, a], []) &\leftarrow \\ p_1([A|X], [B|Y]) &\leftarrow p_1(A, B), p_1(X, Y). \end{aligned}$$

いま、教師に H_1 を提示し等価性質問を行うと、 H_1 は目標の関係を表現するプログラムではないので、反例が返ってくる。ここではその反例を

$$CE_2 = \langle a, a \rangle$$

とする。

CE_2 を用いると、矛盾点追跡により CE_2 が反例である原因をつきとめることができる。いま、 $\langle a, a \rangle$ が負の具体例で、 $\langle [a, b, a], [a] \rangle$ が正の具体例であるにも関わらず、同じ述語記号 p_1 で表現されていることがその原因である。そこで、新たに補助述語 p_2 を導入し、各サンプルタプル $\langle t_1, t_2 \rangle$ に割り当てる述語記号 $\mathcal{P}$ を次のように変更する。

$$\mathcal{P} = \begin{cases} p_1 & (\langle t_1, t_2 \rangle \in R \text{ であるとき}) \\ p_2 & (\text{そうでないとき}) \end{cases}$$

すると、 $\langle t_1, t_2 \rangle \in R$ であるかどうかを問う所属性質問を用いることにより、次のようなテンポラリーモデル I_2 を作成することができる。

$$I_2 = \left\{ \begin{array}{l} p_2(a, a), \\ p_2([b, a], []), \\ p_1([a, b, a], [a]) \end{array} \right\}$$

I_2 と Σ_1 を用いてプログラムを作ると、

$$\begin{aligned} R(X, Y) &\leftarrow p_1(X, Y) \\ p_2(a, a) &\leftarrow \\ p_2([b, a], []) &\leftarrow \\ p_1([A|X], [B|Y]) &\leftarrow p_2(A, B), p_2(X, Y) \end{aligned}$$

という推測プログラム H_2 を得る。

H_2 も、目標の関係を表現するプログラムではないので、反例が返ってくる。その反例を

$$CE_3 = \langle [a|a], [a|a] \rangle$$

とする。 CE_2 の場合と同様にして、今度は $\langle [b, a], [] \rangle$ と $\langle a, a \rangle$ が同じ述語記号 p_2 に割り当てられていることに原因であることがわかる。そこで、述語 p_2 に割り当てられるタプルが $\langle t_1, t_2 \rangle \notin R$ かつ $\langle [a|t_1], [a|t_2] \rangle \in R$ であるものになるようにする。この場合、新述語 p_3 を導入し、タプル $\langle t_1, t_2 \rangle$ に割り当てる述語記号 $\mathcal{P}$ を次のように変更する。

$$\mathcal{P} = \begin{cases} p_1 & (\langle t_1, t_2 \rangle \in R \text{ であるとき}) \\ \mathcal{Q} & (\text{そうでないとき}) \end{cases}$$

$$\mathcal{Q} = \begin{cases} p_2 & (\langle [a|t_1], [a|t_2] \rangle \in R \text{ であるとき}) \\ p_3 & (\text{そうでないとき}) \end{cases}$$

サンプルタプルからテンポラリーモデルを作る際の述語名の割り当て方を表す規則を図1のような述語名決定木で表すことにする。図1において D_1 は I_1 を作ったときの述語名決定木である。 D_2 は I_2 を作ったときのものであり、ノード $R(X, Y)$ は $\mathcal{P}$ を決める条件 $\langle t_1, t_2 \rangle \in R$ を表している。同様に、 D_3 におけるノード $R(X, Y)$ の左の子 $R([a|X], [a|Y])$ は $\mathcal{Q}$ を決める条件 $\langle [a|t_1], [a|t_2] \rangle \in R$ を表している。以降、 CE_4 として $\langle [a, b, a, b, a], [a, b, a] \rangle$ 、 CE_5 として $\langle [b, b, a], [b] \rangle$ が返ってくるとすると、図1の D_3 、 D_4 の述語名決定木が得られ、最終的にテンポラリーモデル

$$I_5 = \left\{ \begin{array}{l} p_1([a, b, a, b, a], [a, b, a]), \\ p_2([b, a, b, a], [b, a]), \\ p_1([a, b, a], [a]), \\ p_2([b, a], []), \\ p_3(a, a), \\ p_4(b, b) \end{array} \right\}$$

を得る。 I_5 を用いれば次の正解のプログラム H_5 が推測として得られる。

$$\begin{aligned} R(X, Y) &\leftarrow p_1(X, Y) \\ p_1([A|X], [B|Y]) &\leftarrow p_3(A, B), p_2(X, Y) \\ p_2([A|X], [B|Y]) &\leftarrow p_4(A, B), p_1(X, Y) \\ p_2([b, a], []) &\leftarrow \\ p_3(a, a) &\leftarrow \\ p_4(b, b) &\leftarrow \end{aligned}$$

まとめると、このアルゴリズムは、推測のプログラムに対して正の反例 $\langle t_1, t_2 \rangle$ が返されると、 $\langle t_1, t_2 \rangle$ の Σ に関する部分タプルをサンプルタプルの集合 S_i に追加して推測のプログラムを更新する。負の反例 $\langle t_1, t_2 \rangle$ が返されれば、 $\langle t_1, t_2 \rangle$ を用いて新述語を導入しサンプルタプルへの述語名の割り当てを変更して、推測のプログラムを更新する。

7.3 準備

一階言語のアルファベットを一つ固定し、 F をその関数記号の集合、 Π を述語記号の集合、 V を変数の集合とする。項の集合を $T(F \cup V)$ 、基礎項の集合を $T(F)$ と表す。 $F|_n$ で F 中の n 引数の関数記号の集合を表す。定数記号の集合は $F|_0$ で表すものとする。 Π についても同様である。 $T(F)^n$ で n 個の $T(F)$ の直積を表し、 $T(F)^* = T(F)^1 \cup T(F)^2 \cup \dots$ と

する。また、 $T(F \cup V)^n$ の要素 $\langle t_1, \dots, t_n \rangle$ を $\mathbf{t}$ と表記することがある。このとき、述語記号 $p \in \Pi_n$ に対して $p(\mathbf{t})$ は原子式 $p(t_1, \dots, t_n)$ を意味する。

$T(F)^n$ の部分集合を **基礎項の n 項関係** という。混乱が生じないときは、単に **関係** という。関係 R に対して、 $\bar{R} = (T(F)^n - R)$ と定義する。 R の要素を R の **正の具体例**、 $\bar{R}$ の要素を R の **負の具体例** という。

本章では論理プログラムを関係の表現手法として扱う。本章の例などで用いる $[A|X]$ などのリストを表す項の表現は Prolog の標準的な記法 [Clocksin&Mellish 94] に従うとする。なお、以降では論理プログラムの確定節を **節**、確定節プログラムを **プログラム** と呼ぶ。また、基礎単位節を **基礎事実**、単位節でない節、すなわち本体を持つ節を **ルール節** という。

関係 R をプログラム P の最小エルブランモデル $M(P)$ を用いて表現するために、 P 中に特別な述語記号 R を用意し、 P には次の形をした R の **定義節** が少なくとも一つ存在することを仮定する。

$$R(X_1, \dots, X_n) \leftarrow p(X_1, \dots, X_n)$$

ここで、 $p \in \Pi_n$ とする。このとき p を **目標述語** とよび、目標述語以外の述語 q を **補助述語** と呼ぶ。 R の定義節を複数用いることによって、 R を P 中のいくつかの目標述語で表される関係の和として表現することも可能である。

関係 R の任意の正の具体例 $\mathbf{t}$ に対して、 $R(\mathbf{t}) \in M(P)$ 、かつ、任意の負の具体例 $\mathbf{s}$ に対して、 $R(\mathbf{s}) \notin M(P)$ であるとき、プログラム P は R を **表現する** という。つまり、学習アルゴリズムの目標は R の正負の具体例の集合から R を表現するプログラム P を見つけることである。

述語変数 $\mathcal{P}$ とタプル $\mathbf{t} \in T(F \cup V)^*$ に対して、 $\mathcal{P}(\mathbf{t})$ を **原子式スキーマ** という。また、 $\mathcal{P}_0, \dots, \mathcal{P}_k$ を互いに異なる述語変数、 $\mathbf{t}_0, \dots, \mathbf{t}_k$ をタプルとすると、

$$\sigma : \mathcal{P}_0(\mathbf{t}_0) \leftarrow \mathcal{P}_1(\mathbf{t}_1), \mathcal{P}_2(\mathbf{t}_2), \dots, \mathcal{P}_k(\mathbf{t}_k)$$

を **節スキーマ** という。特に $k \geq 1$ のとき、 σ を **ルールスキーマ** とよぶ。節スキーマ σ の各述語変数に Π の要素を代入して得られる節 C を σ の **節インスタンス** とよぶ。

定義 タプル $\mathbf{t} \in T(F)^*$ は、 Σ 中のルールスキーマ

$$\sigma : \mathcal{P}_0(\mathbf{u}_0) \leftarrow \mathcal{P}_1(\mathbf{u}_1), \dots, \mathcal{P}_k(\mathbf{u}_k)$$

と $\mathbf{t} = \mathbf{u}_0\theta$ となる θ が存在するとき、 σ を用いて **分解可能** であるという。このとき各 $\mathbf{u}_i\theta$ ($1 \leq i \leq k$) を $\mathbf{t}$ の σ に関する i 番目の **補助タプル** といい、 $Aux_\sigma(\mathbf{t}, i)$ と書く。また、 $\mathbf{t}$ は Σ 中のどの σ を用いても分解可能でないとき、 Σ に関して **分解不可能** であるという。

定義 プログラム P が以下のような3つのプログラム P_{rule} , P_{fact} , P_R の和で表されるとき、 P は Σ のプログラムインスタンスであるという。

1. P_{rule} 中の節は Σ 中のルールスキーマの節インスタンスである。
2. P_{fact} は基礎事実の集合であり、その要素 $p(\mathbf{t}) \leftarrow$ に対して $\mathbf{t}$ は Σ に関して分解不可能である。
3. P_R 中の節は R の定義節である。

Σ のプログラムインスタンスから成るプログラムの族を $\mathcal{C}(\Sigma)$ と定義する。

次に、ルールスキーマ集合が与えられたときの学習問題を定式化する。

定義 R を基礎項の n 項関係とする。ルールスキーマ集合 Σ が与えられたとき、学習アルゴリズム $\mathcal{A}$ が R の具体例の与え方 $\mathcal{E}$ のもとで、 R の Σ に関する複雑さを表すパラメータ m と与えられた具体例の複雑さを表すパラメータ w の多項式時間で R を表現するプログラム $P \in \mathcal{C}(\Sigma)$ を出力するとき、 $\mathcal{A}$ は R をスキーマ Σ のもとで、 $\mathcal{E}$ から効率良く学習するという。

ここで、 R の Σ に関する複雑さを表すパラメータ m とは、 R を $\mathcal{C}(\Sigma)$ 中のプログラム P で表現したときの、 P の複雑さを表すパラメータである。具体例の与え方を $\mathcal{E}$ としては、正の具体例のみをシステムに与え続ける方法 [Shinohara 94]、正の具体例の集合と負の具体例の集合をあらかじめ与えておく方法 [Muggleton 95]、対話的に教師がシステムに与える方法 [Angluin 88, Shapiro 81] などがある。本章では対話的に教師がシステムに与える方法をとる。アルゴリズム $\mathcal{A}$ が R を表現するプログラム P を学習した際、 P に補助述語が含まれている場合は、 $\mathcal{A}$ はその補助述語を発見したことになる。

7.1 節で示したようにスキーマ Σ が与えられたとしても $\mathcal{E}$ が正負の具体例の集合をあらかじめ与えておく方法では、 $P \in \mathcal{C}(\Sigma)$ で表現される関係 R を効率的良く学習することはできない。そこで、学習アルゴリズムが教師に対して R に関する次の二種類の質問をすることを許すことにする。

定義 所属性質問 $MEM(\mathbf{t})$ とは、タプル $\mathbf{t} \in T(\Sigma)^n$ が R の正の具体例であるか負の具体例であるかを質問することである。教師 (ユーザ) は $\mathbf{t}$ が目標の関係 R の正の具体例であるならば *yes*、負の具体例であるならば *no* を返す。等価性質問 $EQUIV(P)$ とは、推測のプログラム P によって表現される関係が R と等しいかどうか質問することである。教師は P が R を表現するプログラムならば *yes* を返し、そうでないとき、反例 $\mathbf{t}$ を返す。ここで、

t が反例であるとは、 t が R の正の具体例かつ $R(t) \notin M(P)$ である(正の反例)か、または、 t が R の負の具体例かつ $R(t) \in M(P)$ である(負の反例)ことをいう。

7.4 正則なルールスキーマ集合を持つ論理プログラム

本章では、正則なルールスキーマ集合というスキーマのクラスを与える。そして、関係 R が正則なルールスキーマ集合 Σ を持つプログラム $P \in \mathcal{C}(\Sigma)$ で表現されるとき、 R と P 、 Σ の間で成り立つ性質を議論する。

定義 ルールスキーマの集合 Σ は次の2つの条件を満たすとき、正則であるという。

1. Σ 中の任意の節スキーマ σ は

$$P_0(t_0) \leftarrow P_1(X_1), \dots, P_k(X_k)$$

という形である。ただし、 X_i は変数のタプル、 t_0 には少なくともひとつの関数記号が現れ、 σ 中に出現する変数は σ の頭部と本体にちょうど一回ずつ現れる。

2. Σ 中の任意の二つのルールスキーマ

$$\sigma_1 : P_0(t_0) \leftarrow P_1(t_1), \dots, P_k(t_k)$$

$$\sigma_2 : Q_0(s_0) \leftarrow Q_1(s_1), \dots, Q_l(s_l)$$

に対して、 t_0 と s_0 は単一化不可能である。

ルールスキーマ集合 Σ が正則であるとき、 $P \in \mathcal{C}(\Sigma)$ は正則なプログラムであるといい、関係 R はある正則なプログラム $P \in \mathcal{C}(\Sigma)$ によって表現されるとき、 Σ に関して正則であるという。

例 7.1 7.2節で用いた Σ_1 は正則なルールスキーマ集合である。

定義 正則なプログラム P が正規形であるとは、任意の P 中の単位節 $p(t)$ について、 t が Σ で分解不可能であることをいう。

ここで、 R を表現する任意の正則なプログラム $P \in \mathcal{C}(\Sigma)$ に対して、 R を表現するある正規形の正則なプログラム $P' \in \mathcal{C}(\Sigma)$ が存在する。

正則な関係 R を表現するプログラム $P \in \mathcal{C}(\Sigma)$ における補助述語を R を用いて特徴づけるために、接続項というタプルを定義する。

定義 正則なルールスキーマ集合 Σ と異なる変数のタプル X に対して Σ 上の X に関する接続項を以下のように帰納的に定義する。

1. X 自身は X に関する接続項である。
2. ルールスキーマ $\sigma \in \Sigma$ の代入例

$$P_0(\alpha) \leftarrow P_1(s_1), \dots, P_{i-1}(s_{i-1}), P_i(\beta), P_{i+1}(s_{i+1}), \dots, P_k(s_k)$$

に対して、 β が X に関する接続項であり、 $s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_k$ が基礎項のタプルであるとき、 α は X に関する接続項である。

3. 以上から接続項と判るものだけが接続項である。

$X = \langle X_1, \dots, X_n \rangle$ に関する接続項 α 、基礎項のタプル $t = \langle t_1, \dots, t_n \rangle$ に対して、 $\alpha.t = \alpha\{X_1/t_1, \dots, X_n/t_n\}$ と定義とする。 X に関する接続項 $\beta = \langle s_1, \dots, s_n \rangle$ に対しては $\alpha.\beta = \alpha'\{Y_1/s_1, \dots, Y_n/s_n\}$ とする。ここで、 α' は α 中の各変数 X_i を新しい変数 Y_i に置き換えたものとする。

例 7.2 7.2節で用いた Σ_1 に対して、 $\alpha = \langle [a|X], [a|Y] \rangle$ は Σ_1 上の $\langle X, Y \rangle$ に関する接続項である。また、 $s = \langle a, a \rangle, t = \langle [b, a], [] \rangle$ 、とすると、 $\alpha.s = \langle [a|a], [a|a] \rangle$ 、 $\alpha.t = \langle [a, b, a], [a] \rangle$ である。

関係 R を表現する正則なプログラム $P \in \mathcal{C}(\Sigma)$ において、 $p(t), p(s) \in M(P)$ というように同じ述語で表される2つの基礎項のタプル t, s を次の同値関係 $\stackrel{R}{\equiv}_{\Sigma}$ を用いて、特徴付けることができる。

定義 R を関係、 Σ をルールスキーマ集合、 i を Σ 中に現れるある原子式スキーマの引数の数とし、タプル $t, s \in T(F)^i$ とする。 $\langle X_1, \dots, X_i \rangle$ に関する Σ 上の任意の深さの接続項 α に対し、 $\alpha.t \in R \iff \alpha.s \in R$ であるとき、 $t \stackrel{R}{\equiv}_{\Sigma} s$ とする。

定理 7.1 ある正則なプログラム $P \in \mathcal{C}(\Sigma)$ で表現される関係 R に対して、 $p(s) \in M(P)$ かつ $p(t) \in M(P)$ ならば、 $s \stackrel{R}{\equiv}_{\Sigma} t$ である。

7.5 学習アルゴリズム

本節では、前節で議論した同値関係 $\stackrel{R}{\equiv}_{\Sigma}$ をもとに、正則なプログラムで表現される関係 R を効率よく学習するアルゴリズムを与える。学習アルゴリズムの概要を図7.2に示す。

Input: 正則なルールスキーマ集合 Σ .

Given: 所属性質問 $MEM(t)$, 等価性質問 $EQUIV(P)$.

Output: プログラム $P \in \mathcal{C}(\Sigma)$.

Procedure $SPIQ(\Sigma)$:

```

1   $S := \phi;$                                 ▷ サンプルタプルの集合.
2   $H := \phi;$                                 ▷ 推測のプログラム.
3   $\mathcal{D}$  を  $\Sigma$  上の初期述語名決定木の集合とする;
4  等価性質問として  $H$  を提示する;
5  repeat
6     $t :=$  反例;
7    if  $R(t) \in M(H)$  then                  ▷ 反例が負.
8       $t$  を用いて述語名決定木の集合  $\mathcal{D}$  を更新する;
9    else                                    ▷ 反例が正.
10      $t$  の  $\Sigma$  に関する部分タプルをすべて  $S$  に追加する;
11  endif;
12   $S$  と  $\mathcal{D}$  からテンポラリーモデル  $I$  を作る;
13   $I$  と  $\Sigma$  から仮説  $H$  を作る;
14   $H$  を提示し, 等価性質問をする;
15  until 等価性質問に対して yes;
16   $H$  を出力して停止する;
end .

```

図 7.2: アルゴリズム SPIQ

7.5.1 サンプルタプルとテンポラリーモデル

定義 タプル t の Σ に関する部分タプルを以下のように帰納的に定義する.

1. t 自身は t の Σ に関する部分タプルである.
2. s が t の Σ に関する部分タプルであり, ある $\sigma \in \Sigma$ と i に関して s が分解可能であるとき, $Aux_\sigma(s, i)$ は t の Σ に関する部分タプルである. ここで, $1 \leq i \leq k$, k はルールスキーマ σ の本体の数とする.
3. 以上のことから部分タプルとわかるものだけが t の Σ に関する部分タプルである.

関係 R とルールスキーマ集合 Σ に対して, $t \in R$ の Σ に関する部分タプル s を (Σ に関する) サンプルタプルと呼ぶ. ルールスキーマ集合 Σ を正則であるとする. R の Σ に関するサンプルタプル t に対して $P \in \mathcal{C}(\Sigma)$ を R を表現する正規形のプログラムとすると, P 中のある述語記号 p に対して, $p(t) \in M(P)$ となる.

サンプルタプルの集合 S に対して, 適当に述語を割り当てて作られる原子式の集合をテンポラリーモデルという. さらに, テンポラリーモデル I に対して, I と Σ を用いて作られる仮説のプログラム $P(I, \Sigma)$ を次のように定義する.

定義 n 項関係 R , ルールスキーマ集合 Σ と原子式の集合 I に対して, プログラム $P(I, \Sigma)$ を基礎事実の集合 P_{fact} , ルール節の集合 P_{rule} と R 定義節の集合 P_R の和と定義する.

$$\begin{aligned}
 P_{fact} &= \{p(t) \in I \mid t \text{ は } \Sigma \text{ に関して分解不可能}\}, \\
 P_{rule} &= \left\{ p_0(t_0) \leftarrow p_1(t_1), \dots, p_k(t_k) \left| \begin{array}{l} \mathcal{P}_0(t_0) \leftarrow \mathcal{P}_1(t_1), \dots, \mathcal{P}_k(t_k) \in \Sigma, \\ \text{代入 } \theta \text{ と } p_0, \dots, p_k \text{ が存在して,} \\ p_i(t_i\theta) \in I (0 \leq i \leq k). \end{array} \right. \right\}, \\
 P_R &= \{R(\mathbf{X}) \leftarrow p(\mathbf{X}) \mid p(t) \in I, t \in R, p \in \Pi|_n, \mathbf{X} \in V^n\}.
 \end{aligned}$$

原子式の集合 I とルールスキーマ集合 Σ から作られるプログラム $P(I, \Sigma)$ について次の定理が成り立つ.

定理 7.2 関係 R , 正則なスキーマ集合 Σ , 原子式の集合 I に対して, $P = P(I, \Sigma)$, $p(s) \in M(P)$, $p(t) \in I$ とする. 任意の $p(u), p(v) \in I$ について $u \stackrel{R}{\equiv}_{\Sigma} v$ ならば, $s \stackrel{R}{\equiv}_{\Sigma} t$ である.

7.5.2 述語名決定木

各サンプルタプル $\langle t_1, \dots, t_i \rangle \in S$ に対して, i 引数の述語名決定木を用いて述語名を割り当てテンポラリモデル I を作る. 述語名決定木は 6.2 節で述べた文字列の状態決定木を基礎項の係に拡張したものである.

定義 Σ 上の n 引数の述語名決定木とは, 葉でない各ノードに原子式 $R(\alpha)$ がラベル付けられていて, 各葉にはそれぞれ異なる述語記号 $p_i \in \Pi|_n$ がラベル付けられている 2 分木のことをいう. ここで, α は Σ 上の $\langle X_1, X_2, \dots, X_n \rangle$ に関する接続項であるとする. 根が葉である述語名決定木を n 引数の初期述語名決定木という.

7.2 節で用いた Σ_1 には 2 引数の原子式スキーマしか現れなかったので, 2 引数の述語名決定木のみを考えた. Σ に異なる引数の原子式スキーマが現れる場合には, 引数 i に対してそれぞれ i 引数の述語名決定木 D^i を用意し, その集合 $\mathcal{D}$ を考える. 以後, Σ に i 引数の原子式スキーマが現れるときには, 述語名決定木の集合 $\mathcal{D}$ には i 引数の述語名決定木がちょうどひとつ含まれているものと仮定する.

定義 $\mathcal{D}$ を述語名決定木の集合, $D^i \in \mathcal{D}$ は i 引数の述語名決定木とする. サンプルタプル $\mathbf{t} \in T(F)^i$ に対して, $\Pi_D^R(\mathbf{t})$ を図 7.3 に示す $\pi_{D^i}^R(\mathbf{t})$ によって求まる原子式 $p(\mathbf{t})$ とする. また, サンプルタプルの集合 S に対して, テンポラリモデル $\Pi_D^R(S)$ を $\{A \mid \mathbf{t} \in S, \Pi_D^R(\mathbf{t}) = A\}$ とする.

学習アルゴリズムは, テンポラリモデル $I = \Pi_D^R(S)$ を用いて推測のプログラム $H = P(I, \Sigma)$ を作り, 教師に等価性質問をする.

7.5.3 述語名決定木の更新

定義 Σ 上の i 引数の述語名決定木 D に対して, p を D に現れる述語記号, α を Σ 上の $\langle X_1, \dots, X_i \rangle$ に関する接続項とする. D の述語 p を α で更新するとは, D の葉 p をノード $R(\alpha)$ で置き換え, $R(\alpha)$ の右の子を p , 左の子を q とすることである. ここで, q は $\Pi|_n$ の要素で, D に現れない新しい述語記号とする.

定義 i 引数の述語名決定木 D^i を含む Σ 上の述語名決定木の集合 $\mathcal{D}$ に対して, $\mathcal{D}$ の述語 $p \in \Pi|_i$ を Σ 上の $\langle X_1, \dots, X_i \rangle$ に関する接続項 α で更新するとは, D^i の述語 p を α で更新することをいう.

次に, 推測のプログラム $H = P(\Pi_D^R(S), \Sigma)$ に対して負の反例が返ってきたときに推測の仮説を修正するアルゴリズムを与える. 負の反例を $\mathbf{s} \in T(F)^n$ とすると, $p(\mathbf{s}) \in M(H)$,

Input: タプル $\mathbf{t} \in T(F)^n$.

Given: i 引数の述語名決定木 D^i , 所属性質問 $MEM(\mathbf{t})$.

Output: 基礎原子式 $p(\mathbf{t})$.

Function $\pi_{D^i}^R(\mathbf{t})$:

```

1  Node :=  $D^i$  の根;
2  while Node が葉でない do
3      Node =  $R(\alpha)$  であるとする;
4      if  $MEM(\alpha.\mathbf{t}) = \text{yes}$  then
5          Node := Node の右の子;
6      else
7          Node := Node の左の子;
8      end if;
9  end ;
10 p := Node;
11 return  $p(\mathbf{t})$ ;
end .
```

図 7.3: i 引数の述語名割り当て $\pi_{D^i}^R$

$p(\mathbf{t}) \in \Pi_D^R(S)$ となる $\mathbf{t} \in R$ が存在する. $\alpha = \langle X_1, \dots, X_n \rangle$ とすると, $\alpha.s \notin R$ かつ $\alpha.t \in R$ である. 定理 7.2 から, S 中のある 2 つのサンプルタプル $\mathbf{u}, \mathbf{v}$ に対して, $\Pi_D^R(\mathbf{u}) = q(\mathbf{u})$, $\Pi_D^R(\mathbf{v}) = q(\mathbf{v})$ かつ $\mathbf{v} \not\stackrel{R}{\sim}_{\Sigma} \mathbf{u}$ である. 定理 7.1 より $\mathbf{u}, \mathbf{v}$ には異なる述語名が割り当てられなければならない. 図 7.4 に示す手続き $FindPred(R(\alpha), p(\mathbf{s}))$ を実行すると $\mathbf{u}, \mathbf{v}$ には異なる述語名を割り当てるように述語名決定木が更新される. なお, $FindPred$ 中にあらわれる $switch_{\Sigma}(\mathbf{s}, i, \mathbf{t})$ と $diff_{\Sigma}(\mathbf{s}, i, \mathbf{t})$ を以下のように定義する.

定義 Σ を正則なルールスキーマ集合とし, 基礎項のタプル $\mathbf{s}, \mathbf{t}$ は Σ 中のスキーマ $\sigma = \mathcal{P}_0(\mathbf{u}) \leftarrow \mathcal{P}_1(\mathbf{u}_1), \dots, \mathcal{P}_k(\mathbf{u}_k)$ を用いて分解可能であるとする. σ の代入例

$$\mathcal{P}_0(\mathbf{s}) \leftarrow \mathcal{P}_1(\mathbf{s}_1), \dots, \mathcal{P}_k(\mathbf{s}_k),$$

$$\mathcal{P}_0(\mathbf{t}) \leftarrow \mathcal{P}_1(\mathbf{t}_1), \dots, \mathcal{P}_k(\mathbf{t}_k)$$

に対して,

$$\mathcal{P}_0(\mathbf{v}) \leftarrow \mathcal{P}_1(\mathbf{s}_1), \dots, \mathcal{P}_i(\mathbf{s}_i), \mathcal{P}_{i+1}(\mathbf{t}_{i+1}), \dots, \mathcal{P}_k(\mathbf{t}_k)$$

が σ の代入例となるような基礎項の組 $\mathbf{v}$ を $switch_{\Sigma}(\mathbf{s}, i, \mathbf{t})$ と表す. ただし, $0 \leq i \leq k$ とする. また, $switch_{\Sigma}(\mathbf{s}, 0, \mathbf{t}) = \mathbf{t}$, $switch_{\Sigma}(\mathbf{s}, k, \mathbf{t}) = \mathbf{s}$ である. さらに,

$$\mathcal{P}_0(\alpha) \leftarrow \mathcal{P}_1(\mathbf{s}_1), \dots, \mathcal{P}_{i-1}(\mathbf{s}_{i-1}), \mathcal{P}_i(\mathbf{X}), \mathcal{P}_{i+1}(\mathbf{t}_{i+1}), \dots, \mathcal{P}_k(\mathbf{t}_k)$$

が σ の代入例となるような接続項 α を $diff_{\Sigma}(\mathbf{s}, i, \mathbf{t})$ と表す. ここでは, $1 \leq i \leq k$ とする.

例 7.3 7.2 節で用いた反例 $\mathbf{s} = CE_3$ を考える.

$$FindPred(R(X, Y), p_1(\langle [a|a], [a|a] \rangle))$$

を実行して, $\mathbf{t} = \langle [a, b, a], [a] \rangle$ が選ばれたとする. 所属性質問を用いると,

$$MEM(\mathbf{t}) = yes$$

$$MEM(switch(\mathbf{s}, 1, \mathbf{t})) = MEM(\langle [a, b, a], [a] \rangle) = yes$$

$$MEM(switch(\mathbf{s}, 2, \mathbf{t})) = MEM(\langle [a|a], [a|a] \rangle) = no$$

である. さらに

$$diff(\mathbf{s}, 2, \mathbf{t}) = \langle [a|X], [a|Y] \rangle, \mathbf{s}_2 = \langle a, a \rangle$$

であるので,

$$FindPred(R(\langle [a|X], [a|Y] \rangle), p_2(a, a))$$

Input: 原子式 $R(\alpha)$, $p(\mathbf{s})$.

Given: 述語名決定木の集合 $\mathcal{D}$, 所属性質問 $MEM(\mathbf{t})$.

Output: 述語名決定木の集合 $\mathcal{D}'$.

Procedure $FindPred(R(\alpha), p(\mathbf{s}))$:

```

1   $p(\mathbf{s}) \leftarrow p_1(\mathbf{s}_1), \dots, p_k(\mathbf{s}_k)$  が  $P$  中の節  $C$  の代入例かつ,
2   $p_1(\mathbf{s}_1), \dots, p_k(\mathbf{s}_k) \in M(P)$  となる節  $C \in P$  を見つける;
3   $p(\mathbf{t}) \leftarrow p_1(\mathbf{t}_1), \dots, p_k(\mathbf{t}_k)$  が  $C$  の具体例になる  $p(\mathbf{t}), p_1(\mathbf{t}_1), \dots, p_k(\mathbf{t}_k) \in I$  を見つける;
4  if  $MEM(\alpha.t) = no$  then
5       $\mathcal{D}$  の述語  $p$  を  $\alpha$  で更新し,  $\mathcal{D}'$  とする;
6      output  $\mathcal{D}'$ ;
7  else
8      for  $i = 1$  to  $k$  do
9           $\mathbf{v}_i := switch(\mathbf{s}, i, \mathbf{t})$ ;
10         if  $MEM(\mathbf{v}_i) = no$  then
11              $\alpha' := diff(\mathbf{s}, i, \mathbf{t})$ ;
12              $FindPred(R(\alpha.\alpha'), p_i(\mathbf{s}_i))$ ;
13         end if;
14     end for;
15 end if;
end .
```

図 7.4: 手続き $FindPred$

が実行され、 $MEM([a|a], [a|a]) = no$ であるので、述語 p_2 が $R([a|X], [a|Y])$ で更新される。その結果、同じ述語名 p_2 が割り当てられていたサンプルタプル $\langle [b, a], [] \rangle$, $\langle a, a \rangle$ に別々の述語名が割り当る。

手続き *FindPred* の着想は矛盾点追跡 [Shapiro 81] にある。しかし、補助述語に関する質問を用いない点で Shapiro の実現したアルゴリズムとは異なっている。また、矛盾点追跡から生まれた Ishizaka のアルゴリズム [Ishizaka 89, Ishizaka 90] とも異なっている。Ishizaka のアルゴリズムはあらかじめ多めに仮説となる可能性がある節をすべて加えておいて間違っている節を見つけ削除する。一方、本章のアルゴリズムは最小限の節を仮説に加え、間違っている述語の割り当て方を見つけだし、新述語を導入してそれを修正する。

7.5.4 サンプルタプルの追加

推測のプログラム $H = P(\Pi_D^R(S), \Sigma)$ に対して正の反例 t が返ってきた場合には、その t の Σ に関する部分タプルをすべてサンプルタプルの集合 S に追加する。

7.5.5 本章の主定理

定理 7.3 正則なルールスキーマ集合 Σ に対して、 R をプログラム $P \in \mathcal{C}(\Sigma)$ で表現される関係とする。学習アルゴリズム *SPIQ* は R を Σ のもとで、所属性質問と等価性質問から効率良く学習する。

証明 定理を証明するために、サンプルタプルの集合 S に対して次を定義する。

定義 Σ を正則なルールスキーマ集合、 R を基礎項の n 項関係とする。原子式の集合 I が任意の $p(s), q(t) \in I$ について、 $p \neq q$ ならば $s \not\equiv_{\Sigma}^R t$ を満たすとき、 I は $\equiv_{\Sigma}^R$ に関して無矛盾であるという。

計算量の評価のためにサンプルタプル t の Σ に関する大きさ $|t|_{\Sigma}$ を t の Σ に関する部分タプルの数と定義する。補題 7.3 より学習アルゴリズムがメインループを繰り返している間、サンプルタプルの集合 S は常に $\equiv_{\Sigma}^R$ に関して無矛盾である。補題 7.4 より、正の反例が返るたびに推測のプログラムの節の数が増える。また、明らかに負の反例が返るたびに推測のプログラムの述語数が増える。

補題 7.5 と 7.6 より、推測のプログラム H の節数と述語数はそれぞれ、 R を表現するあるプログラム $P \in \mathcal{C}(\Sigma)$ の節数 m を越えることはない。よって学習アルゴリズムは停止する。

反例が返ってくる回数は、高々 $2m$ 回である。したがって、等価性質問の回数は高々 $2m$ 回である。また、述語名決定木の深さは高々 m である。

反例として返される各タプル t に対し、 $|t|_{\Sigma}$ の最大値を w とするとサンプルタプルの集合 S の要素数は、 wm を越えることはない。

正の反例が来たときに新しくサンプルタプルを追加するのに必要な計算量は、 $O(mw)$ であり、ルールスキーマ集合に含まれる節スキーマの本体の数の最大値を k とすると、負の反例が来たときに更新すべき述語をみつける計算量は $O(kw)$ である。述語名決定木の更新の計算量は $O(m)$ 、推測のプログラムを作る計算量は $O(mw)$ である。所属性質問の回数は高々 $m(mw + (k+1)w)$ である。したがって、学習アルゴリズムは、関係 R を表現するプログラム $P \in \mathcal{C}(\Sigma)$ の節数 m と反例の大きさの最大値 w の多項式時間で R を学習できる。□

7.6 定理 7.1 の証明

証明 正則なプログラムの定義より、 P の本体中には関数記号が現れず、変数 X は本体中にちょうど 1 回しか出現しない。したがって、 Σ 上の $\langle X_1, \dots, X_n \rangle$ に関する任意の接続項 α に対して、 $p(\alpha.s) \in M(P)$ ならば $p(\alpha.t) \in M(P)$ である。よって、 $R(\alpha.s) \in M(P)$ ならば $R(\alpha.t) \in M(P)$ である。□

7.7 定理 7.2 の証明

証明 $p(s) \in T_P \uparrow d$ とし、 d に関する数学的帰納法による。 $d = 1$ のとき、節 $p(s) \leftarrow P$ に含まれる。 $P = P(I, \Sigma)$ の作り方より、 $p(s) \in I$ であり、仮定より $s \equiv_{\Sigma}^R t$ であるから定理は成り立つ。次に $p(s) \in T_P \uparrow d$ なる任意の $p(s) \in M(P)$ について定理が成り立つと仮定し、 $p(s') \in T_P \uparrow d+1$ とする。 $p(s') \leftarrow p_1(s'_1), \dots, p_k(s'_k)$ が P 中の節 C の代入例で、 $p_i(s'_i) \in T_P \uparrow d$ となる $p_i(s'_i)$ と節 C が存在する。ところで、 P の作り方から I 中に $p(u) \leftarrow p_1(u_1), \dots, p_k(u_k)$ が C の代入例となる原子式 $p(u), p_1(u_1), \dots, p_k(u_k)$ が存在する。ここで、帰納法の仮定より、任意の $1 \leq i \leq k$ について $s'_i \equiv_{\Sigma}^R u_i$ であり、下に示す補題 7.2 から $s' \equiv_{\Sigma}^R u$ 。さらに、定理の仮定から任意の $t \in I$ について $t \equiv_{\Sigma}^R u$ である。よって、任意の $t \in I$ について $s' \equiv_{\Sigma}^R t$ となり $d+1$ のときも成り立つ。□

7.7.1 定理7.2に関する補題

補題 7.1 Σ を正則なルールスキーマ集合とする. α を Σ に関する接続項とする. $s \stackrel{R}{\equiv}_{\Sigma} t$ ならば, $\alpha.s \stackrel{R}{\equiv}_{\Sigma} \alpha.t$ である.

証明 接続項 β に対して, $\beta.(\alpha.s) \in R$ とする. $\alpha' = \beta.\alpha$ とすると, 仮定より, $\alpha'.t \in R$ である. よって, $\beta.(\alpha.t) \in R$. $\beta.(\alpha.t) = (\beta.\alpha).t$ であるので, $\alpha.s \stackrel{R}{\equiv}_{\Sigma} \alpha.t$ である. $\square$

補題 7.2 R を関係, Σ を正則なスキーマ集合とする. s, t がともに $\sigma \in \Sigma$ を用いて分解可能であり, σ の本体の数は k とする. 任意の $i = 1, 2, \dots, k$ について $Aux_{\sigma}(s, i) \stackrel{R}{\equiv}_{\Sigma} Aux_{\sigma}(t, i)$ ならば, $s \stackrel{R}{\equiv}_{\Sigma} t$ である.

証明 任意の $i = 1, 2, \dots, k$ について, $\alpha_i = \text{diff}_{\Sigma}(s, i, t)$, $t_i = Aux_{\sigma}(t, i)$, $s_i = Aux_{\sigma}(s, i)$ とおく. 補題 7.1 と仮定より, $\alpha_i.t_i \stackrel{R}{\equiv}_{\Sigma} \alpha_i.s_i$ が成り立つ. ここで $\alpha_i.t_i = \text{switch}(s, i-1, t)$, $\alpha_i.s_i = \text{switch}(s, i, t)$ であるので, $\text{switch}(s, i-1, t) \stackrel{R}{\equiv}_{\Sigma} \text{switch}(s, i, t)$ である. よって, $t = \text{switch}(s, 0, t) \stackrel{R}{\equiv}_{\Sigma} \text{switch}(s, 1, t) \stackrel{R}{\equiv}_{\Sigma} \dots \stackrel{R}{\equiv}_{\Sigma} \text{switch}(s, k, t) = s$ となる. $\stackrel{R}{\equiv}_{\Sigma}$ は同値関係なので $s \stackrel{R}{\equiv}_{\Sigma} t$ である. $\square$

7.8 定理7.3に関する補題

補題 7.3 $\mathcal{D}$ を Σ 上の述語名決定木の集合, S をサンプルタブルの集合, Σ を正則なルールスキーマ集合とする. すると, 原子式の集合 $\Pi_D^R(S)$ は $\stackrel{R}{\equiv}_{\Sigma}$ に無矛盾である.

証明 任意の $s, t \in S$ について, $\Pi_D^R(s) = p(s)$, $\Pi_D^R(t) = q(s)$ かつ $p \neq q$ ならば $s \not\stackrel{R}{\equiv}_{\Sigma} t$ である. $\square$

補題 7.4 Σ を正則なルールスキーマ集合, $\mathcal{D}$ を Σ 上の任意の述語名決定木の集合とする. 推測プログラム $H = P(\Pi_D^R(S), \Sigma)$ に対して, 正の反例が返ってきたとする. すると, つぎの推測プログラム H' の節数は H より少なくともひとつ増える.

証明 正の反例 t の Σ に関する部分タブルの集合 S_t に対して, H_t を $P(\Pi_D^R(S_t), \Sigma)$ とする. $H' = H \cup H_t$ である. 明らかに, $R(t) \in M(H_t)$ であり, $H_t \subseteq H$ と仮定すると $R(t) \in M(H_t)$ であり t が正の反例であることに矛盾. $\square$

補題 7.5 原子式の集合 I が正則なルールスキーマ集合 Σ 上の同値関係 $\stackrel{R}{\equiv}_{\Sigma}$ に関して無矛盾であり, $p(t) \in I$ ならばある Σ に関する接続項 α に対して $R(\alpha.t)$ が正の具体例となるとする. $P' \in \mathcal{C}(\Sigma)$ を R を表す Σ に関する正規形のプログラムであるならば P' の節数は $P(I, \Sigma)$ の述語数以上である.

証明 $p(t), q(t) \in I$, $p \neq q$ とする. 定理 7.1 より, P' において $p'(t) \in M(P')$, $q'(t) \in M(P')$ かつ $p' \neq q'$ である. よって, P' の述語数は $P(I, \Sigma)$ の述語数以上である. P' の述語数は P' の節数以下であると仮定してもかまわないので補題は成り立つ. $\square$

補題 7.6 原子式の集合 I が正則なルールスキーマ集合 Σ 上の同値関係 $\stackrel{R}{\equiv}_{\Sigma}$ に関して無矛盾であり, $p(t) \in I$ ならばある Σ に関する接続項 α に対して $R(\alpha.t)$ が正の具体例となるとする. $P' \in \mathcal{C}(\Sigma)$ を R を表す Σ に関する正規形のプログラムであるならば P' の節数は $P(I, \Sigma)$ の節数以上である.

証明 $P(I, \Sigma)$ 中に

$$p_0(u_0) \leftarrow p_1(u_1), \dots, p_k(u_k)$$

という節 C が存在すると仮定する ($k \geq 0$). $P(I, \Sigma)$ の作り方から, ある基礎代入 θ が存在し, $p_0(u_0\theta), p_1(u_1\theta), \dots, p_k(u_k\theta) \in I$ である. 仮定より, ある Σ に関する接続項 α に対して $R(\alpha.u_0\theta)$ が R の正の具体例となる. したがって, P' において, ある述語 p'_0 が存在し, $p'_0(u_0\theta) \in M(P')$ である. $P' \in \mathcal{C}(\Sigma)$ より, P' 中に

$$p'_0(u_0\theta) \leftarrow p'_1(u_1\theta), \dots, p'_k(u_k\theta)$$

がその基礎代入例となるような節 C' が存在する. また, $P(I, \Sigma)$ 中に

$$p_0(u_0) \leftarrow p_1(u_1), \dots, p_k(u_k)$$

$$q_0(u_0) \leftarrow q_1(u_1), \dots, q_k(u_k)$$

という2つの節 C が存在し, ある i ($0 \leq i \leq k$) に対して, $p_i \neq q_i$ とする. I の作り方より, ある基礎代入 θ_1, θ_2 が存在し, $p_0(u_0\theta_1), p_1(u_1\theta_1), \dots, p_k(u_k\theta_1) \in I$, $q_0(u_0\theta_2), q_1(u_1\theta_2), \dots, q_k(u_k\theta_2) \in I$ である. 仮定より, ある Σ に関する接続項 α, β に対して $R(\alpha.(u_i\theta_1))$, $R(\beta.(u_i\theta_2))$ がともに R の正の具体例となる. よって, P' 中のある述語 p'_i, q'_i について $p'_i(u_i\theta_1)$, $q'_i(u_i\theta_2) \in M(P')$ となる. $u_i\theta_1 \not\stackrel{R}{\equiv}_{\Sigma} u_i\theta_2$ であり, 定理 7.1 から, $p'_i \neq q'_i$ である. $\square$

7章のまとめ

正則なルールスキーマ集合 Σ が与えられたとき，所属性質問と等価性質問を用いることにより，補助述語を発見しながら基礎項の n 項関係を効率良く学習する手法を提案した．ここで，スキーマとは論理プログラムの変数の呼び出し方を限定し，対象となる仮説空間を制限するものである．提案した学習アルゴリズムは述語名決定木を用いた構成的なものである．正則な Σ に対して， $\mathcal{C}(\Sigma)$ 中の論理プログラム P で表される木の関係は， P 中の節の数 m と与えられた反例の大きさ w の多項式時間で学習可能であることを証明した．その結果，これまで知られていた有限オートマトンや木オートマトンの学習手法 [Angluin 87, Takada 88, Sakakibara 90] を質問とスキーマを併用した帰納論理プログラミングとして一般化した．

この論文ではスキーマを所与のものと仮定しており，それ自体を生成する問題は未解決である．この問題に対しては，既知のプログラムからスキーマを取りだして利用する手法が考えられる．例えば，記号 a, b を用いた有限オートマトンを一つでも知っていれば，それから記号 a, b 上のスキーマを作ることが可能である．この手法は類推の分野で広く研究され，既に学習への応用も行われている [Sadohara 95, Sadohara 97]．本論文のアルゴリズムを適用することにより，さらに強力な類推システムの構築が可能となるであろう．

第8章

結論

本論文では，帰納論理プログラミングにおける以下の二つの課題を解決した．

[課題1] 学習や発見の仕組み，すなわち，理論的背景が明らかなシステムを構成する．

[課題2] 既に計算論的学習理論の分野で研究されてきた学習方式との関係を明らかにする．

最初の課題に対して，底節交差汎化法という手法を提案した．底節交差汎化法は SOLDR 導出・ボトムアップ評価・包摂に基づく共通汎化という三つの単純な手続きを組み合わせた手法である．底節交差汎化法によって，相対包摂に基づく共通汎化をすべて求めることができる．従来の主要な帰納論理プログラミング手法も相対包摂に基づく共通汎化を求めるものであったが，理論的な整備の遅れから，扱うことができる規則に大きな制限があった．本論文では，底節交差汎化法を提案することによりこの制限を取り払い，より広いクラスに対する帰納論理プログラミングへの道を切り開いた．

二つ目の課題に対して，帰納論理プログラミングにおける補助述語の自動生成問題を正則言語の学習問題に関連づけた．補助述語の自動生成は広い探索空間を必要とする困難な問題として知られている．従来から，プログラムのスキーマを与えるなどの手法が提案されていた．スキーマとは生成される仮説や学習対象となるプログラムに対する強い構文的な制約を与えるものである．本論文では，たとえ学習の対象となる論理プログラムのスキーマを与えたとしても，所与の正負の具体例からの学習は非効率的であることを証明した．そこで，質問とスキーマを併用する新たな手法を提案し，同手法によって効率的な学習が可能となるクラスを明らかにした．

本章では，これら本論文で得られた結果についてまとめる．

8.1 底節交差汎化法

底節交差汎化法は、背景知識となる論理プログラムと具体例となる複数個の確定節から、具体例を説明する仮説を計算する手法である。

同手法は、SOLDR 導出・ボトムアップ評価・包摂に基づく共通汎化という三つの手法に基づいている。SOLDR 導出は、論理プログラムを用いた発想推論 (アブダクション) の実現手法である。ボトムアップ評価 [Ullman 88] は演繹データベースの分野で開発された演繹推論 (デダクション) の手法である。包摂に基づく共通汎化 [Plotkin 70] は帰納論理プログラミングの研究分野で開発された一般化の手法である。これら三つの手法に基づいて仮説を生成することで、同手法による仮説生成の過程を論理的に意味付けた。

また、底節交差汎化法で求まる仮説の特徴付けを、相対包摂に基づく共通汎化 [Plotkin 71] によって与えた。すなわち、同手法で求まる仮説はすべて与えられた具体例の相対包摂に基づく共通汎化である。逆に、与えられた具体例の相対包摂に基づく共通汎化は、すべて底節交差汎化法によって求めることができる。従来の主要な帰納論理プログラミング手法 [Muggleton 95] も相対包摂に基づく共通汎化を求めるものであったが、理論的な整備の遅れから、扱うことができる規則には大きな制限があった。本論文では、底節交差汎化法を提案することによりこの制限を取り払った。

底節交差汎化法に関連する理論的研究として RLGG [Buntine 88] や $CIG_{\geq \Sigma}$ [Page&Frish 92] が挙げられる。両者とも述語記号が異なる具体例を扱えない。それに対し、底節交差汎化法は述語記号が異なる具体例を扱うことができる。

8.2 質問とスキーマを用いた述語発見による論理プログラムの学習手法

正則なルールスキーマ集合 Σ が与えられたとき、所属性質問と等価性質問を用いることにより、補助述語を発見しながら基礎項の n 項関係を効率良く学習する手法を提案した。ここで、スキーマとは論理プログラムの変数の呼び出し方を限定し、対象となる仮説空間を制限するものである。その結果、これまで知られていた有限オートマトンや木オートマトンの学習手法 [Angluin 87, Takada 88, Sakakibara 90] を質問とスキーマを併用した帰納論理プログラミングとして一般化した。

一般に多項式時間学習アルゴリズムは枚挙のための仮説空間の大きさが多項式で抑えられることを示すにとどまることが多い。それに対して、本論文では、述語名決定木を用いた構成的な学習アルゴリズムを与え、その実行時間が多項式で抑えられることを証明した。

この論文ではスキーマを所与のものと仮定しており、それ自体を生成する問題は未解決である。この問題に対しては、既知のプログラムからスキーマを取りだして利用する手法が考えられる。例えば、記号 a, b を用いた有限オートマトンを一つでも知っていれば、それから記号 a, b 上のスキーマを作ることが可能である。この手法は類推の分野で広く研究され、既に学習への応用も行われている [Sadohara 95, Sadohara 97]。本論文のアルゴリズムを適用することにより、さらに強力な類推システムの構築が可能となるであろう。

謝辞

本論文をまとめるにあたって御世話になった多くの方々に心から感謝の意を申し上げたい。

本論文の主査である原口誠教授をはじめ、情報エレクトロニクス系の諸先生方には日頃から親身な御指導を頂いた。原口誠教授には本研究を仕上げるにあたり大変お世話になった。先生からは研究に関してだけでなく研究会での心構えなど多くの親身な御助言を頂いた。田中譲教授にはゼミや普段の活動を通して、ときには優しくときには厳しく研究に関する様々な事柄を親身になって指導して頂いた。宮本衛市教授には研究発表に関する重要な注意点を頂いた。山本章博助教授には研究室に入ったころからずっと御世話になりっぱなしであった。先生は多忙な毎日にもかかわらず、私を書く舌足らずの資料や論文の草稿をいつも丹念に読んで下さり、適切な指導をして下さった。

アラン・ランザー助手、客員研究員のベルトラム・フロンヘッファー氏には英語発表をする際に大変お世話になった。佐藤健助教授には Space Collaboration System(SCS) を用いたテレビ会議での発表を御世話して頂いた。大久保好章助手には日頃から様々な面で御世話になった。二つ上の先輩である助手の赤石美奈先生と何人かの学生で行った二年間のデータベースゼミは週一回の輪読であったが大変勉強になった。

また、九州大学の有川節夫教授、九州工業大学の篠原武教授、東京大学の宮野悟教授、慶應大学の古川康一教授、九州大学の有村博紀助教授、篠原歩助教授、電子技術総合研究所の佐土原健博士をはじめ計算論的学習理論研究グループの諸先生方からは、私が研究をはじめてまもない学部の学生であったころから、学会、セミナー、SCS などその他様々な機会を通して、多くの御助言を頂いた。

知識メディアラボトリの学生のみなさんにも大変お世話になった。平成11年3月に博士課程を修了した石野明君とは学部の学生であった頃から一緒であった。彼は計算機全般にわたって大変詳しく、どんな些細な質問に対してもいつも明快な答えを返してくれた。彼とは興味の対象が一致することが多く、いつもお互いにアイデアを出し合っていた気がする。UNIX 管理に詳しい平野亮太君は忙しい中、UNIX 上で起こる様々なトラブルをいつも

すぐに解決してくれ、私のとるに足りない質問にも快く答えてくれた。また、明るい後輩たちに囲まれて充実した学生生活を送ることができたことにも感謝したい。このほか、大学以外のたくさんの方々からも大変暖かい励ましを頂いた。

末筆ながら、長期にわたる学生生活を辛抱強く支えてくれた帯広の両親には最大の感謝を申し上げたい。

参考文献

- [相笠 99] 相笠健一: 帰納論理プログラミングに基づく知識メディアの合成システム, 北海道大学工学部電子工学科卒業論文 (1999)
- [Angluin 87] Angluin, D.: Learning Regular Sets from Queries and Counter-Examples, *Information and Computation*, Vol. 75, pp.87-106 (1988).
- [Angluin 88] Angluin, D.: Queries and Concept Learning, *Machine Learning*, Vol. 2, pp.319-42 (1988).
- [有川&宮野 86] 有川節夫, 宮野悟: オートマトンと計算可能性, 培風館 (1986).
- [有川&原口 88] 有川節夫, 原口誠: 述語論理と論理プログラミング, オーム社 (1988).
- [有川&西野 88] 有川節夫, 西野哲朗: 学習における計算論的アプローチ-機械学習に向けて-, 情報処理, Vol.32, No.3, pp.217-25 (1991).
- [Arikawa et. 92] Arikawa, S., Shinohara, T., Yamamoto, A.: Learning elementary formal systems, *Theoretical Computer Science*, Vol.95, No.1, pp.97-113 (1992).
- [Arimura 97] Arimura, H.: Learning Acyclic First-order Horn Sentences From Implication, In *Proceedings of the 8th International Workshop on Algorithmic Learning Theory (LNAI 1316)*, pp.432-45 (1997).
- [Brainerd 68] Brainerd, W.S.: The Minimalization of Tree Automata, *Information and Control*, vol.13, pp.484-491 (1968).
- [Buntine 88] Buntine, W.L.: Generalized Subsumption and Its Application to Induction and Redundancy, *Artificial Intelligence*, Vol.36, pp.149-76 (1988).
- [Clocksin&Mellish 94] Clocksin, W.F. and Mellish, C.S.: *Programming in Prolog*, fourth edition, Springer (1994).
- [Colmerauer et. 73] Colmerauer, A., Kanoui, H., Roussel, P. and Pasero, R.: Un Système de Communication Homme-Machine en Français, Research Report. France: Université d'Aix-Marseille II, Groupe de Recherche en Intelligence Artificielle (1973).

- [Flach 93] Flach, P.A.: Predicate Invention in Inductive Data Engineering, *Proc. of ECML-93. European Conference on Machine Learning (LNAI 667)*, pp.83-94 (1993).
- [Flach 94] Flach, P.: *Simply Logical - Intelligent Reasoning by Example -*, Wiley (1994).
- [Flener 96] Flener, P.: Inductive Logic Program Synthesis with DIALOGS, *Proc. of the 6th International Workshop on Inductive Logic Programming, ILP-96 (LNAI 1314)*, pp.175-98 (1997).
- [古川 97] 古川康一: 帰納論理プログラミング-チュートリアル-, 人工知能学会誌, vol.12, No.5, pp.655-64 (1997).
- [Genesereth 93] Genesereth, M.R., Nilsson, N.J. (古川康一 監訳): 人工知能基礎論, オーム社 (1993).
- [Gold 67] Gold, E.M.: Language Identification in the Limit, *Information and Control*, Vol. 10, pp.447-474 (1967).
- [Gold 78] Gold, E.M.: Complexity of Automaton Identification from Given Data, *Information and Control*, Vol. 37, pp. 302-20 (1978).
- [Herbrand 30] Herbrand, J.: Recherches sur la Théorie de la Démonstration, *Travaux de la Société des Sciences et de Lettres de Varsovie, Classe III Sci. Math. Phys.*, 33 (1930)
- [Hirata&Haraguchi et. 99] Hirata, A., Haraguchi, M. and Sadohara, K.: On Moded Functional IP Syntheses by Reusing Comporsition Structures, *Workshop on Applied Learning Theory*, DOI Technical Report, pp.59-69 (1998)
- [Hirowatari 94] Hirowatari, E. and Arikawa, S.: Incorporating Explanation-Based Generalization with Analogical Reasoning, *Bulletin of Informatics and Cybernetics*, Vol. 26, No.1-2, pp.13-33 (1994).
- [ICOT 95] モデル生成型定理証明系: MGTP マニュアル, 新世代コンピュータ技術開発機構 (1995).
- [Inoue 92a] Inoue, K.: Linear Resolution for Consequence Finding, *Artificial Intelligence*, Vol.56, pp.301-53 (1992).
- [井上 92b] 井上克己: アブダクションの原理, 人工知能学会誌, Vol.7, No.1, pp.48-59 (1992).
- [Ishizaka 89] Ishizaka, H.: Inductive Inference of Regular Languages based on Model Inference, *Int. J. Comput. Math.*, Vol. 27, No.2, pp.67-83 (1989).
- [石坂 89] 石坂裕毅: 述語生成に関する一考察, *Technical Memorandum TM-0631*, ICOT (1989).

- [Ishizaka 90] Ishizaka, H.: Polynomial Time Learnability of Simple Deterministic Languages, *Machine Learning*, Vol. 5, No.2, pp.151-64 (1990).
- [Ito&Yamamoto 92] Ito, K., Yamamoto, A.: Polynomial-time MAT Learning of Multilinear Logic Programs, *Proc. of Third Workshop on Algorithmic Learning Theory (LNAI 743)*, pp.63-74 (1993).
- [Ito&Yamamoto 98] Ito, K., Yamamoto, A.: Finding Hypotheses from Examples by Computing the Least Generalization of Bottom Clauses, In *Proceedings of the First International Conference on Discovery Science (LNAI 1532)*, pp.303-14 (1998)
- [Jung 93] Jung, B.: On Inverting Generality Relations, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pp.87-101 (1993)
- [Kearns&Vazirani 94] Kearns, Y., Vazirani, U.: *An Introduction to Computational Learning Theory*, The MIT Press (1994).
- [キッスイリクン, 沼尾&志村 92] ブンサーム・キッスイリクン, 沼尾正行, 志村正道: 弁別に基づく構成的帰納学習, 人工知能学会誌, Vol. 7, No.6, pp.1027-37 (1992).
- [Kleene 52] Kleene, S.C.: Finite axiomatizability of theories in the predicate calculus using additional predicate symbols, *Two Papers on the Predicate Calculus*, No.10, Memoirs of American Mathematical Society (1952)
- [Kowalski 74] Kowalski, R.A.: Predicate Logic as a Programming Language, *Information Processing 1974*, pp.569-574 (1974).
- [Kowalski 79a] Kowalski, R.A.: *Logic for Problem Solving*, Elsevier North-Holland, New York (1979). (浦昭二監修, 山田真市, 菊地光昭, 桑野龍夫共訳: 論理による問題の解法, 倍風館)
- [Kowalski 79b] Kowalski, R.A.: Algorithm=Logic+Control, *Communications of the Association for Computing Machinery*, Vol. 22, No. 4, pp.424-36 (1979).
- [Lloyd 83] Lloyd, J.W.: An Introduction to Deductive Database Systems, *Australian Computer Journal*, Vol. 15, No. 2, (1983).
- [Lloyd 84] Lloyd, J.W.: *Foundations of Logic Programming*, Springer-Verlag (1984).
- [Marriott 88] Marriott, K., Naish, L., and Lassez, J.L.: Most Specific Logic Programs, In *Logic Programming Proceedings of the Fifth International Conference and Symposium*, pp.910-923, MIT Press (1988).
- [Mitchell 86] Mitchell, T. M., Keller, R.M. and Kedar-Cabelli, S.T.: Explanation-Based Generalization A Unifying View, *Machine Learning*, Vol. 1, No.1, pp.47-80 (1986)

- [溝口 97] 溝口文雄: 帰納論理プログラムの適用方法について, 人工知能学会誌, vol.12, No.5, pp.675-82 (1997).
- [Mofizur&Numao 94] Mofizur, C.R., Numao, M.: Constructive Induction for Recursive Programs, *Proc. of 4th International Workshop on Analogical and Inductive Inference, 5th International Workshop on Algorithmic Learning Theory (LNAI 872)*, pp.161-75 (1994).
- [Moore 56] Moore, E.F.: Gadanken-experiments on Sequential Machines, *Automata Studies*, Shannon, C.E. and Macasthy, J. ed. pp.129-153 (1956).
- [Muggleton 91] Muggleton, S.: Inductive Logic Programming, *New Generation Computing*, Vol.8, No.4, 295-318 (1991).
- [Muggleton 94] Muggleton, S., De Raedt, L.: Inductive Logic Programming: Theory and Methods *J. of Logic Programming*, Vol. 19-20, pp.629-79 (1994).
- [Muggleton 95] Muggleton, S.: Inverse Entailment and Prolog *New Generation Computing*, Vol. 13, No.3-4, pp.245-86 (1995).
- [Muggleton et.98] Muggleton, S., Srinivasan, A., King, R.D., Sternberg M.J.E.: Biochemical Knowledge Discovery Using Inductive Logic Programming, In *Proceedings of the First International Conference on Discovery Science (LNAI 1532)*, pp.326-41 (1998).
- [Nienhuys-Cheng& Wolf 97] Nienhuys-Cheng, S., Wolf, R.D.: *Foundations of Inductive Logic Programming*, Springer (1997).
- [長崎&田中 94] 長崎祥, 田中譲: シンセティック・メディアシステム IntelligentPad, コンピュータソフトウェア, Vol. 11, No. 1, pp.36-48 (1994).
- [沼尾 97] 沼尾正行: 帰納論理プログラムの応用-近年の国際会議の動向を踏まえて-, 人工知能学会誌, Vol. 12, No.5, pp.683-88 (1997).
- [大須賀&佐伯 87] 大須賀節夫, 佐伯胖: 知識の獲得と学習, オーム社 (1987).
- [Page&Frisch 92] Page, C.D., Jr., Frisch, A.M.: Generalization and Learnability: A Study of Constrained Atoms, *Inductive Logic Programming*, Muggleton, S.(ed.), Academic Press, pp.29-61 (1992).
- [Peirce] Peirce, C.S: *Collected Papers of Charles Sanders Peirce (1839-1914)* Hartshorne, C.S, Weiss, P.(eds.), The Belknap Press (1965).
- [Plotkin 70] Plotkin, G. D.: A Note on Inductive Generalization, *Machine Intelligence*, Vol.5, pp. 153-63 (1970).

- [Plotkin 71] Plotkin, G. D.: A Further Note on Inductive Generalization, *Machine Intelligence*, Vol.6, pp. 101-24 (1971).
- [Quinlan 96] Quinlan, J.R. : Learning first-order definitions of functions *J. Artif. Intell. Res.*, Vol.5 (1996).
- [Rissanen 78] Rissanen, J.: Modelling by Shortest Description, *Automatica*, Vol.14, pp. 465-71 (1978).
- [Raedt&Bruynooghe 92] De Raedt, L., Bruynooghe, M. : Interactive Concept-Learning and Constructive Induction by Analogy, *Machine Learning*, Vol. 8, No.2, pp.107-50 (1992).
- [Robinson 65] Robinson, J.A.: A Machine-oriented Logic Based on the Resolution Principle, *JACM*, Vol. 12, No.1, pp.23-41 (1965).
- [Roussel 75] Roussel, P.: PROLOG: Manuel de Reference et d'Utilization, Groupe de Recherche en Intelligence Artificielle (1975).
- [Rouveirol 92] Rouveirol, C.: Extensions of Inversion of Resolution Applied to Theory Completion, *Inductive Logic Programming*, Muggleton, S.(ed.), Academic Press, pp.63-92 (1992).
- [Sadohara 95] Sadohara, K., Haraguchi, M.: Analogical Logic Program Synthesis from Examples, *Proc. 8th European Conference on Machine Learning (LNAI 912)*, pp.232-44 (1995).
- [Sadohara 97] Sadohara, K., Haraguchi, M.: Using Abstraction Schemata in Inductive Logic Programming, *Proc. of 7th International Workshop on Inductive Logic Programming (LNAI 1297)*, pp.189-204 (1997).
- [Sakakibara 90] Sakakibara, Y.: Inductive Inference of Logic Programs Based on Algebraic Semantics, *New Generation Computing*, Vol. 7, pp.365-80 (1990).
- [Shapiro 81] Shapiro, E.Y.: Inductive Inference of Theories From Facts, Technical Report 192, Yale University (1981). Reprinted in: *Computational Logic*, Lassez, J.L. and Plotkin, G.(ed.), pp.199-254, The MIT Press (1991). 有川節夫 (訳): 知識の帰納的推論, 共立出版 (1986).
- [Shinohara 94] Shinohara, T.: Rich Classes Inferable from Positive Data: Length-bounded Elementary Formal Systems, *Information and Computation*, Vol. 108, No.2, pp.175-86 (1994).
- [SICStus 96] the Intelligent Systems Laboratory: SICStus Prolog User's Manual. Swedish Institute of Computer Science (1996).

- [Stahl 95] Stahl, I.: The Appropriateness of Predicate Invention as Bias Shift Operation in ILP, *Machine Learning*, Vol. 20, No.1-2, pp.95-117 (1995).
- [Sterling&Shapiro 86] Sterling, L. and Shapiro, E. : *The Art of Prolog*, The MIT Press, Cambridge MA (1986).
- [Takada 88] Takada, Y.: Gramatical Inference for Even Linear Languages Based on Control Sets, *Information Processing Letters* Vol.28, pp.193-99 (1988)
- [Ullman 88] Ullman, J.D. : *Database and Knowledge-base Systems*, Computer Science Press, Rockville MD (1988).
- [Wirth&O'Roke 91] Wirth, R., O'Rourke, P.: Constrains on Predicate Invention, *Proc. of the Eighth International Workshop on Machine Learning (ML91)*, pp.457-61 (1991).
- [Yamamoto 89] Yamamoto, A.: Short Note: Procedural Semantics and Negative Information of Elementary Formal System *Journal of Logic Programming*, Vol.13, No.1, pp.89-97 (1992).
- [山本 97] 山本章博: 帰納論理プログラムの基礎理論, 人工知能学会誌, vol.12, No.5, pp.665-74 (1997).
- [Yamamoto 97a] Yamamoto, A.: Representing Inductive Inference with SOLD-Resolution, The extended abstract is in *Proceedings of the IJCAI'97 Workshop on Abduction and Induction in AI*, pp.59-63 (1997).
- [Yamamoto 97b] Yamamoto, A.: Which Hypotheses Can Be Found with Inverse Entailment?, In *Proceedings of the Seventh International Workshop on Inductive Logic Programming (LNAI 1297)*, pp.296-308 (1997).
- [Yamamoto 98a] Yamamoto, A.: An Inference Method for the Complete Inverse of Relative Subsumption, *New Generation Computing* Vol. 17, No. 1, pp.99-117 (1999).
- [Yamamoto 98b] A. Yamamoto. Revising the Logical Foundations of Inductive Logic Programming Systems with Ground Reduced Programs. *New Generation Computing* Vol. 17, No. 1, pp.119-127 (1999).
- [Yamamoto 98c] A. Yamamoto. Using Abduction for Induction based on Bottom Generalization, To appear in A. Kakas and P. Flach (eds.) *Abductive and Inductive Reasoning : Essays on their Relation and Integration* (1998).

付録 A

ILP システム・BORDA の全ソースプログラム

```
%%%----- main.pl -----%%%
```

```
%%% 節 C1 と節 C2 を相対包摂する節を求める.
```

```
bottomReduction(E1,E2,B,H):-
```

```
    botc(E1,B,C),
```

```
    botc(E2,B,D),
```

```
%%    same_head_pred(C,D),
```

```
    lgg(C,D,H).
```

```
%%% 底汎化法による仮説の生成
```

```
botc(E,B,[CHead, :-|CBody]):-
```

```
    skolemize(E,[EHead, :-|EBody]),
```

```
    add(EBody,B,P),!,
```

```
    soldr([EHead],P,[CHead]),
```

```
    lm(P,CBody).
```

```
%%    unskolemize(,).
```

```
%% SOLD-導出
```

```
soldr([],_Prog,[]).
```

```
soldr([G|Gs],Prog,F):-
```

```
    member(C,Prog),
```

```
    copy_term(C,[G, :-|Bs]),
```

```
    append(Bs,Gs,NewG),
```

```
    soldr(NewG,Prog,F).
```

```
%ゴールが空であるとき終了
```

```
%選択された原子式が P 中の
```

```
%ルール of の頭部と単一化可能
```



```

solder([G|Gs],Prog,[G]):-
    solder(Gs,Prog,[]).
solder([F|Gs],Prog,[F]):-      % 選択された原子式が P 中の節の
    solder(Gs,Prog,[F]).      % 頭部と単一化できない Abd に追加

%%%----- add.pl -----%%%

%%% 背景知識への単位節の追加

add(EBody,BK,Prog):-
    disj2conj(EBody,Atoms),
    append(Atoms,BK,Prog).

%%% disj2conj([A,B,C],[[A],[B],[C]]).
disj2conj([],[]).
disj2conj([A|X],[[A,:-|Y]]:-
    disj2conj(X,Y).

%%%----- models.pl -----%%%

%%% ボトムアップでモデルをもとめるプログラム
%%%
%%% | ?- models([[p(a)],[p(b)],[p(f(X)),:-,p(X)],3,A).
%%%
%%% A = [p(a),p(b),p(f(a)),p(f(b)),p(f(f(a))),p(f(f(b)))] ?

lm(Prog,Atoms):-tp(Prog,10,Atoms).

%%% tp(Program,Number,Atoms).
tp(_,0,[]):-!.
tp(Program,N,Atoms):-
    N1 is N-1,
    tp(Program,N1,Atoms1),
    tp_operator(Program,Atoms1,Atoms).

%%% tp_operator(Program,InputAtoms,OutputAtoms)
tp_operator(Program,InputAtoms,OutputAtoms):-
    findall(Atom,program_cover(InputAtoms,Program,Atom),RedundantAtoms),

```

```

    remove_duplicates(RedundantAtoms,OutputAtoms).

%%% program_cover(Atoms,Program,Atom)
program_cover(Atoms,Program,Atom):-
    copy_term(Program,Program2),
    member(Cluase,Program2),
    clause_cover(Atoms,Cluase,Atom).

%%% clause_cover(Atoms,Cluase,Atom)

clause_cover(Atoms,[Head,:-|Body],Head):-
    subset(Body,Atoms).

sample_program([[p(a),:-],[p(f(X)),:-,p(X)]]).

%%% 部分集合
%%% subset(+,+).
subset([],_).
subset([A|X],Y):-
    member(A,Y),
    subset(X,Y).

%%%----- skolem.pl -----%%%
%%% 節をスコールム化する

skolemize(C,SK):-
    copy_term(C,SK),
    term_variables(SK,List),
    varlist_skolemize(List).

varlist_skolemize([]).

varlist_skolemize([A|X]):-
    sk_new(SK),
    A=SK,
    varlist_skolemize(X).

```



```
%% スコーレム定数を戻す. (スコーレム定数をずらして Generalize)
```

```
unskolemize(SK,C):-
    skshift(SK,SK2),
    term_subsumer(SK,SK2,C).
```

```
%% 節をスコーレム化する
```

```
:-abolish(sk_next).
:-assert(sk_next(sk1)).
```

```
sk_new(X):-
    sk_next(X),
    retract(sk_next(X)),
    sk_getNext(X,Y),
    assert(sk_next(Y)).
```

```
sk_getNext(X,Y):-
    name(X,[115,107|Char]),
    name(N,Char),
    N1 is N+1,
    name(N1,Char1),
    name(Y,[115,107|Char1]).
```

```
%% スコーレム定数をずらす.
```

```
skshift(S,SS):-
    nonvar(S),
    \+compound(S),
    name(S,SN),
    SN=[115,107|_],
    sk_getNext(S,SS),!.
```

```
skshift(S,S):-
    nonvar(S),
    \+compound(S).
```

```
skshift(X,X):- var(X).
```

```
list_skshfit([],[]).
```

```
list_skshfit([SA|X],[A|Y]):-
```

```
skshift(SA,A),
list_skshfit(X,Y).
```

```
skshift(Comp,X):- compound(Comp),
    Comp=..[F|Arg],
    list_skshfit(Arg,Arg1),
    X=..[F|Arg1].
```

```
%%%----- num.pl -----%%%
num(X):-my_number(S),num_henkan(S,X).
```

```
my_number(0).
my_number(suc(X)):-
    my_number(X).
```

```
num_henkan(0,0).
num_henkan(suc(S),Y):-
    num_henkan(S,X),
    Y is X+1.
```

```
%%%----- gen.pl -----%%%
%% 2つの節を一般化する.
:-use_module(library(terms)).
:-use_module(library(lists)).
```

```
lgg(C,D,X):-clause_thetaGeneralise(C,D,X).
```

```
%% 節の一般化
```

```
clause_thetaGeneralise([C1],[C2|_],X):-
    atom_generalize(C1,C2,X).
clause_thetaGeneralise([C1|_],[C2],X):-
    atom_generalize(C1,C2,X).
clause_thetaGeneralise([H1, :-|B1],[H2, :-|B2],G):-
    atom_generalize(H1,H2,_Atom),
    list_product(B1,B2,C1),
    list_iterate(B1,B2,C2),
    term_generalise([H1, :-|C1],[H2, :-|C2],X),
    clause_removePredVar(X,G).
```



```
%%% リストからの原子式のみを取ってくる。(述語変数を削除)
```

```
clause_removePredVar([], []).
clause_removePredVar([A|X], Y):-
    var(A),!,
    clause_removePredVar(X,Y).
clause_removePredVar([A|X], [A|Y]):-
    clause_removePredVar(X,Y).
```

```
%%% リストの数をそろえる.
```

```
%%% list_product([a,b,c],[1,2],[a,a,b,b,c,c]).
```

```
%%%
```

```
list_product([],_X,[]).
list_product([A|X],Y,Z):-
    atom_product(A,Y,P),
    list_product(X,Y,Q),
    append(P,Q,Z).
```

```
%%% リストの数をそろえる.
```

```
%%% list_iterate([a,b,c],[1,2],[1,2,1,2,1,2]).
```

```
%%%
```

```
list_iterate([],_X,[]).
list_iterate([_A|X],Y,Z):-
    list_iterate(X,Y,Q),
    append(Y,Q,Z).
```

```
atom_product(_A,[],[]).
```

```
atom_product(A,[_B|X],[A|Z]):-
    atom_product(A,X,Z).
```

```
%%% 項の一般化
```

```
term_generalise(X,Y,Z):-
    term_subsumer(X,Y,Z).
```

```
%%% 原子式の一般化
```

```
atom_generalize(X,Y,Z):-
    X=..[Pred|TupleX],
    Y=..[Pred|TupleY],
    term_generalise(TupleX,TupleY,TupleZ),
```

```
Z=..[Pred|TupleZ].
```

```
%%%----- conective.pl -----%%%
```

```
%%% 頭部と接続がある本体を選んだ節
:-consult(num).
```

```
clause_conective([Head],[Head]).
clause_conective([H, :-|B],[H, :-|S]):-
    bodySelect(H,B,S),!.
```

```
%%% 頭部と接続がある本体を選んだリスト
```

```
bodySelect(Head,List,Set):-
    num(N),
    conectNset(Head,List,N,Set),
    N1 is N+1,
    conectNset(Head,List,N1,Set1),
    Set1==Set,
    !.
```

```
%%% 頭部と長さ N で接続がある本体のリスト
```

```
conectNset(_P,_List,0,[]):-!.
conectNset(P,List,N,Z):-
    setof(A,(conectN(P,List,N,A)),Z),!.
conectNset(_P,_List,_N,[]):-!.
```

```
%%% 頭部と長さ N で接続がある本体
```

```
conectN(P,List,1,A):-!,
    member(A,List),
    conective(P,A).
conectN(P,List,N,A):-
    N1 is N-1,
    conectN(P,List,N1,B),
    member(A,List),
    conective(B,A).
```

```
%%% 頭部と直接接続がある本体
```

```
conective(P,Q):-
```



```

term_variables(P,VP),
term_variables(Q,VQ),
member(A,VQ),
member(B,VP),
A==B,!.
```

```

%%%----- init.pl -----%%%
:-use_module(library(lists)).
:-use_module(library(terms)).

:-consult(main).
:-consult(gen).
:-consult(skolem).
:-consult(add).
:-consult(models).
:-consult(conective).

borda(C,D,BK,H):-bottomReduction(C,D,BK,X),clause_conective(X,H).
```