

解説



知識処理分野における自動プログラミング としての機械学習†

篠原 靖志†† 矢澤 利弘†† 原田 実†††

1. はじめに

人間は「～の状況ならば～と判断できる」という形の知識を使った判断をしばしば行っている。知識処理システム（特に、エキスパートシステム）では、このような判断規則を自動的に組み合わせることで、人間なみの高度な判断を実現しようとしている。このために、知識処理システムでは、FORTRAN や C など手続き型言語ではなく、モジュール性の高い決定表、決定木、プロダクションルール、論理式などに基づいた言語を使うことが多い。特に、プロダクションルールや一階述語論理（または、そのサブセットや拡張）に基づく論理プログラム（PROLOG など）は、エキスパートシステムなどでよく使用される。

知識処理分野で用いられるこれらのプログラムに対する自動プログラミングの研究は、「知識＝プログラム」という立場から、知識の自動獲得、すなわち、機械学習の研究として行われてきている。本解説では、このような研究について、特に、そこで用いられている学習方式を中心に解説する。

2. 機械学習

ここでは、基本的には、機械学習を、いくつかの事実が与えられたときに、それらの事実と一致するプログラム P を求める問題として捉える。すなわち、正しい事実とは両立し、誤った事実とは両立しないプログラム P を求める問題である。

プログラム P が事実 p と両立するとは、 P と $\sim p$ が

矛盾することである（ $P \& \sim p = \text{偽}$ ）。たとえば、プログラム $\{a \rightarrow b, a\}$ は、 a および b と両立するが、 $\sim a$ 、 $\sim b$ は両立しない。また、入力 a に対して出力 b を出す関数（プログラム） P は、入出力が (a, b) であるという事実と両立するが、入出力が (a, c) であるという事実とは両立しない。プログラム P が事実 p と両立することを $P \vdash p$ 、両立しないことを $P \nvdash p$ と書くことにする。また、（その真偽は問わず）与えられる可能性のある事実 $(a, b, \sim a, \sim b)$ や入出力 (a, c) など）を問題とよぶ。その全体を問題空間 Problem、実際に与えられた問題を例題とよぶことにする。

自動生成されるプログラムに要求される性質として、事実との両立性のほかに、プログラムの簡単さや効率の良さなどがある。与えられる事実の真偽が正しくない場合もあり、このときには、事実との両立性よりも、プログラムの簡単さなどのほうが重視される場合もある。

そこで、機械学習システムを、以下のような自動プログラミングシステムとして定義する。

『いくつかの例題 p_i とその真偽 t_i の組 $\langle p_i, t_i \rangle$ を入力として、それらの例題を含む問題の集合 C と、それに属する（ほとんど）すべての問題を正しく（効率的に）解く（簡単な）プログラム P を出力するシステムである。』

機械学習システム L :

$$(\text{Problem} \times \{\text{true}, \text{false}\})^N \rightarrow 2^{\text{Problem}} \times \text{Program}$$

$$(\langle p_1, t_1 \rangle, \dots, \langle p_n, t_n \rangle) \rightarrow (C, P)$$

ただし（ほとんど）すべての p_i について

$$t_i = \text{true} \text{ ならば } P \vdash p_i$$

$$t_i = \text{false} \text{ ならば } P \nvdash p_i$$

前述のように、簡単なプログラムであること（プログラムサイズが小さいなど）を重視するか、効率の良いプログラムであることを重視するかなど、プログラム P のどの性質を重視するかは、扱う問題の種類や学習の目的により異なる。なお、 $\langle p_i, t_i \rangle$ で $t_i = \text{true}$ で

† Machine Learning as Automatic Programming in Knowledge Engineering by Yasuaki SHINOHARA, Toshihiro YAZAWA (Central Research Institute of Electric Power Industry, Economic Research Center, Information System Department, Knowledge Engineering Section) and Minoru HARADA (Aoyama Gakuin University, Faculty of Science and Engineering).

†† (財)電力中央研究所経済研究所情報システム部知識処理研究室

††† 青山学院大学理工学部経営工学科

ある例 p_i を正の例, $t_i = \text{false}$ である例 p_i を負の例とよぶ。

このとき、機械学習システムを規定するためには、生成されるプログラムの全体 Program (プログラム空間, 規則空間とよぶ), および, それらが扱う問題の全体 Problem (問題空間, 事例空間とよぶ) をどう設定するかが問題となる。さらに, 実際に学習システムに例題を与えながら学習を進めていくには, どのような性質の例題群 ($\langle p_1, t_1 \rangle, \dots, \langle p_n, t_n \rangle$) がどの程度の数, どのように与えられるかも問題となる。たとえば, 例題集合が同時にすべて与えられる場合と, 逐次与えられる場合とでは機械学習の方式が異なってくる。また, 正の例のみが与えられるのか, 負の例も与えられるのかについても機械学習システムの能力を左右するポイントである。あるいは, 与えられる例の真偽 t_i が正しくない可能性があるのかないのかも大きな影響がある。機械学習は, いくつかのタイプに分類できる。本稿では, そのうち帰納学習, 演繹学習を中心に述べる。

3. 帰納学習によるプログラムの生成

帰納学習は, 提示された (複数の) 例題の間にある一般的な性質, 法則性を見つける学習であり, 論理的能力 (competence) の学習といえる。「例からのプログラミング」は, その代表例である。さまざまな症状の有無とその病名という事例が分かっているとき, 新しい患者の診断のため, この症状があればこの病気だというような診断規則, 診断プログラムを生成する問題も, その一例である。

帰納学習は, 提示された (複数の) 例題を正しく解くことのできるプログラムで, かつ, 一般性の高いものをプログラム空間からみつけた探索の問題として定式化できる。したがって, 一番単純な帰納学習のアルゴリズムは, プログラム空間中のすべてのプログラムを数え上げて, 先の条件を満たすものを検索する「数え上げアルゴリズム」 (生成検査アルゴリズム; Generating and Testing) である。しかし, 可能なプログラムは一般に多数存在し, 単純な数え上げアルゴリズムでは非効率である。このため, より効率的なプログラム空間の探索が必要となる。本章では, 帰納学習をプログラム空間の探索の問題として捉え, この立場からいくつかの学習システムについて述べる。

3.1 プログラム空間の探索による学習

[一般性に基づくプログラム空間の構造]

プログラム空間内のプログラムの間には, その論理

的能力の大小を定義できる。すなわち, 「プログラム A と両立する問題 p が, すべてのプログラム B と両立する ($A \vdash p$ ならば $B \vdash p$) 場合, プログラム B はプログラム A より「一般的」である (以後, B) A と書く)。このとき, プログラム A はプログラム B より「特殊」である ($A \ll B$) ともいう。たとえば,

$$\{p \leftarrow q, p \leftarrow r\} \gg \{p \leftarrow q\} \gg \{p \leftarrow q \& r\}$$

$$\{p \leftarrow q, q\} \gg \{p, q\} \gg \{p\} \gg \{p \vee q\}$$

$$\forall x, p(x) \gg p(a) \gg \exists x, p(x)$$

$$f(x) = x (x \in \text{実数}) \gg f(n) = n (n \in \text{整数})$$

が成立する。この一般性についてプログラム空間は擬順序集合となる⁴⁾。

[一般性に基づくプログラム空間の構造の例]

たとえば, 植物の病気の種類を判定する問題を考えてみる。この問題では, 斑の有無 (A), 虫食い (B), 発育不良 (C), 根枯れ (D) から病気かどうかを判定する。このとき, 病気の判定が, 斑の有無 (A), 虫食い (B), 発育不良 (C), 根枯れ (D) の論理積のみで判断できると仮定する。すなわち, 許されるプログラムが, 「病気 $\leftarrow A, B, C, D$ 中の任意個の論理積」という形のホーン節一つのみの論理式とする。このとき, プログラム空間は, $\{\text{病気} \leftarrow X\} \mid X \text{ は, } A, B, C, D \text{ 中の任意個の論理積}\}$ となる。このプログラム空間には, 図-1 に示す 16 個のプログラムがあり, 「一般性」に関する束をなしている。

[一般化規則によるプログラム空間の定義]

プログラムを以上の形の論理積のみによる形式に限定した場合, 図-1 の例からも分かるように, プログラム空間は条件削除規則: $P \leftarrow X \& Y \Rightarrow P \leftarrow X$ が張る空間になっている。なぜなら, $(P \leftarrow X \& Y) \vdash p$ ならば $(P \leftarrow X) \vdash p$ であるから, 条件削除規則によって, 左辺の論理式の and 条件が削除することでより「一般的な」論理式を得ることができる。このような規則を一般化規則 (Generalization Rule) という。一般化規則には, このほかに, 定数変数化規則 ($P(c) \Rightarrow \forall X, P(X)$), 内遷言付加規則 ($P \leftarrow X \& Y \Rightarrow P \leftarrow X \vee Y$), 遷言付加規則 ($P \leftarrow X \Rightarrow P \leftarrow X \vee Y$, すなわち, $P \leftarrow X \Rightarrow P \leftarrow X, P \leftarrow Y$), 新述語を付加する規則 ($X \Rightarrow X, Y$) などがある⁵⁾。

一般に, プログラム空間全体を枚挙によって定義することは非効率的であり, このようになんらかの一般化規則が張る (束) 空間として構成的にプログラム空間とその「一般性」についての構造を定義することが多い。

A: 斑の有無 B: 虫食い C: 発育不良 D: 根枯れ とする.

Problem = {病気 $\leftarrow$ X | X=A, $\bar{A}$, B, $\bar{B}$, C, $\bar{C}$, D, $\bar{D}$, の任意個の連言 (AND 結合)}

Program = {病気 $\leftarrow$ X | X=A, B, C, D, の任意個の連言 (AND 結合)}

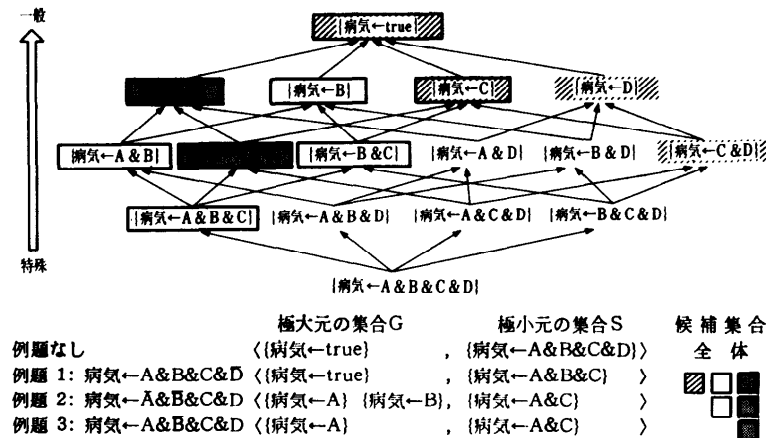


図-1 一般性についての半順序に基づくプログラムの学習

【一般性に基づくプログラム空間の探索】

いま、与えられた例題の真偽 t_i には誤りがないと仮定する。このとき、求めるプログラムはすべての例題を正しく解くことができるものでなければならない。このとき、帰納学習を行ううえで重要な二つの性質が一般性の定義より導かれる。

【性質1】 正の例 e が与えられた ($\langle e, \text{true} \rangle$) のとき、

求めるプログラム A は例題 e と両立するプログラムでなければならないから、求めるプログラム A は、例題 e のみと両立するプログラム E より「一般的な」プログラムである。

($E \vdash e$) & ($p \neq e$ ならば $E \not\vdash p$) のとき、 $A \gg E$

【性質2】 負の例 e が与えられた ($\langle e, \text{false} \rangle$) のとき、

求めるプログラム A は例題 e と両立するプログラムであってはならない。したがって、求めるプログラム A は、例題 e と両立するプログラム E より「一般的な」プログラムではない。

$E \vdash e$ のとき、 $\text{not } A \gg E$

この二つの性質を用いて、逐次提示される例題により、プログラム空間を絞り込んで、最終的に適切なプログラムを求めることができる。

【一般性に基づくプログラム空間の探索の例】

前述の病気診断の例では、例題がない状態では、学習結果となるプログラムの候補の集合 (候補集合) はプログラム空間全体に等しい。ここで、斑 (A)、虫食い (B) があり、発育不良 (C) であるが、根枯れはしていない ($\bar{D}$) のとき、病気であるという例題 1 $\langle \text{病気} \leftarrow A \& B \& C \& D, \text{true} \rangle$ が観測されると、候補集合はこの

観察事実のみと両立するプログラム $\{\text{病気} \leftarrow A \& B \& C\}$ より「一般的」なプログラムの集合に絞り込まれる。

さらに、例題 2 $\langle \text{病気} \leftarrow \bar{A} \& B \& C \& D, \text{false} \rangle$ が観測されると、候補集合からプログラム $\{\text{病気} \leftarrow C \& D\}$ より「一般的」なプログラムは除かれる。さらに、例題 3 $\langle \text{病気} \leftarrow A \& B \& C \& D, \text{true} \rangle$ が観測されると、候補集合はさらに絞り込まれて、 $\{\text{病気} \leftarrow A\}$ 、 $\{\text{病気} \leftarrow A \& C\}$ となる。すなわち、図-1 に示したプログラム空間のみが許されるときには、例題 1、例題 2、例題 3 から「斑があり、発育が悪いならば病気である ($\{\text{病気} \leftarrow A \& C\}$)」ことが確実に分かる。「斑があれば病気である ($\{\text{病気} \leftarrow A\}$)」というプログラムも否定できない。このように、候補集合中の一般性に関する極小元 ($\{\text{病気} \leftarrow A \& B\}$) はもっとも保守的な候補を示し、極大元 ($\{\text{病気} \leftarrow A\}$) はもっとも楽観的な候補となる。以下では、この候補集合の一般性に関する極大元の集合を G 、極小元の集合を S と書く。

【候補集合の保持】

このようなプログラム空間の探索を行う場合、候補集合をそのままつづければ、多くの記憶領域が必要となる。しかし、候補集合は一般性について束を成すので、極大元の集合 G と極小元の集合 S のみによって規定できる。後述する LEX で使われているバージョン空間法⁴⁾では、 $\langle G, S \rangle$ によって候補集合 (バージョン空間とよぶ) を保持し、新しい例題 p のみと両立するプログラム P と G, S から新しい候補集合 $\langle G', S' \rangle$ を求めている。また、MSC 一般化 (most specific conjunction generalization)⁶⁾ といわれる問題などの

楽観的な学習結果のみが必要な場合は G のみ、保守的な学習結果のみが必要な場合は S のみを保持、更新すればよい。

3.2 探索の効率化

【探索高速化の一般的手法：正の例、負の例の一般性】

上の病気診断の例では、正の例だけでは、楽観的な候補集合 G はまったく更新されないし、負の例だけでは、保守的な候補集合 S はまったく更新されない。正の例によると、主に保守的な仮説集合 S が絞り込まれ、負の例によると、主に楽観的な仮説集合 G が絞り込まれる。正の例と負の例を共に活用することが、効果的なプログラム空間の絞り込みには重要である。

また、プログラム空間の絞り込みを大きくするためには、例題に対応するプログラムの一般化（正の例）、特殊化（負の例）も重要である。

プログラム空間における枝刈りの条件【性質 1】は、次のように拡張される。

【性質 1'】の正の例 e と両立するプログラム E に対し、「 E と両立する問題 p は、すべて正の例である」（ E は正の例のための十分条件である）場合、求めるプログラム A はプログラム E より「一般的な」プログラムである。

（ $E \vdash p$ ならば、 $\langle p, \text{true} \rangle$ ）のとき、 $A \gg E$

したがって、【性質 1'】から、保守的な候補集合 S から効率的に候補集合を絞り込むには、正例に対する十分条件であるプログラム E で一般性の高いプログラムを見つければよい。

一方、【性質 2】から、楽観的な候補集合 G から効率的に候補集合を絞り込むには、負の例題 e と両立するプログラム P で特殊性の高いプログラムを見つければよい。

【LEX：正の例に対応するプログラムの一般化による刈込みの効率化】

Mitchell らの LEX⁷⁾ は、記号積分の基本演算子（部分積分公式、和の積分公式など）をどのような形の式に適用すると効率的に積分ができるかを示すヒューリスティックな計算公式（ $\int X \text{ 三角関数}(X) dX \rightarrow$ 部分積分 $uv - \int v du [u = X, dv = \text{三角関数}(X) dX]$ など）を生成するシステムである。各演算子を使う式の形の記述（ヒューリスティックの左辺）には、具体的な式（ $\int 1 dx, \int \sin(x) dx$ など）から任意の関数の積分を表す記述（ $\int f dG$ ）までさまざまな抽象度の記述が考えられる。LEX ではこれらの記述の間の一般性に関

する束をプログラムの文法規則としてあらかじめもっている。LEX では、積分の計算公式を基本的に次の①②の繰り返しによって求めている。

① 問題解決器により基本演算を使って、具体的な式を積分してみる。

② その成功／不成功から正の例／負の例を作成して、バージョン空間法によって各演算子を適用する条件を絞り込む。

この繰り返しによって、最終的に候補集合 $\langle G, S \rangle$ に残った唯一の要素が求める計算公式である。

たとえば、 $\int x(\cos(x) + 2x) dx$ の積分を試みると、部分積分の仕方によって積分の成功／不成功が異なる。

$$\int x(\cos(x) + 2x) dx$$

→〈部分積分 $[u = x, dv = (\cos(x) + 2x) dx]$ 〉→

$$uv - \int v du = x(\sin(x) + x^2) + \int (\sin(x) + x^2) dx$$

→〈和の積分〉→

$$x(\sin(x) + x^2) + \int \sin(x) dx + \int x^2 dx$$

→〈正弦関数の積分公式〉→

$$x(\sin(x) + x^2) - \cos(x) + \int x^2 dx + C$$

→〈 $x^n (n \neq -1)$ の積分公式〉→

$$x(\sin(x) + x^2) - \cos(x) + x^3/3 + C \quad \text{【成功】}$$

$$\int x(\cos(x) + 2x) dx$$

→〈部分積分 $[u = \cos(x) + 2x, v = x dx]$ 〉→

$$uv - \int u dv = (\cos(x) + 2x)x^2/2 +$$

$$\int (-\sin(x) + 2) d(x^2/2)$$

→〈…〉→

...

【失敗】

この結果、部分積分の適用についての正の例の一つとして

$$\int x(\cos(x) + 2x) dx$$

→部分積分 $[u = x, dv = (\cos(x) + 2x) dx]$

...正の例 I

が、また、負の例の一つとして、

$$\int x(\cos(x) + 2x) dx$$

→部分積分 $[u = \cos(x) + 2x, dv = x dx]$

...負の例 I

が得られる。これらの正/負の例によって部分積分 ($\int u dv \rightarrow uv - \int v du$) を適用する式の形 (例題がない場合は, $\int F dG$ より特殊な式の全体) を絞り込む。

LEX では、候補集合の極小元の集合 S を効果的に絞り込むために、4. で述べる説明による学習 (Explanation-Based Learning) で使われる制約後向き伝播手続き (Constraint Back-Propagation Procedure) によって、[性質 1'] の E に相当するプログラムを求めることが提案されて

いる⁷⁾。すなわち、ある例題について基本演算子列 $OP_1, \dots, OP_n$ が積分に成功した場合、その基本演算子列が積分に成功する (すなわち、積分を含まない式になる) ための十分条件を求める。これは、 $(OP_n \dots OP_1)^{-1}$ (積分を含まない式の全体) を表す式の記述を求めることになる。たとえば、上の例題では、 x^n についての積分公式が $n \neq -1$ に適用可能なことから、 $\int x(\cos(x) + nx^{n-1})dx (n \neq -1)$ が十分条件となる。したがって、

$$\int x(\cos(x) + nx^{n-1})dx (n \neq -1)$$

→部分積分 [$u=x, dv=(\cos(x)+nx^{n-1})dx$]

…正の例 II

が証明される。この正の例 II は正の例 I より「一般的」で、かつ、[性質 1'] の E の条件を満たすプログラムであるから候補空間 $\langle G, S \rangle$ を正の例 I より効果的に絞り込むことができる (図-2(a))。

[MIS: 負の例に対応するプログラムの特殊化による刈込みの効率化]

Shapiro の MIS (Model Inference System)⁸⁾ は、一階述語論理 (特に、ホーン節 ($P \leftarrow Q_1, \dots, Q_n$) 集合) を対象とした帰納学習システムである。MIS では、一般化規則の代りに、精密化演算子 (refinement operator) を使ってプログラム空間を定義する。「精密化」の概念 (正しくは、その逆の概念) は、「一般性」の概念とプログラムの「簡単さ」の概念を併せたもので、節 (clause) に適用される。「節 A が節 B より精密」ということは「節 B が節 A より一般的 ($\{A\} \vdash p$ ならば $\{B\} \vdash p$)」で、かつ、節 B の複雑さが節 A の複雑さより小さい ($B \leq pA; \leq p$ は節の複雑さに関する半順序関係)」ということである。精密化演算子のみが張る空

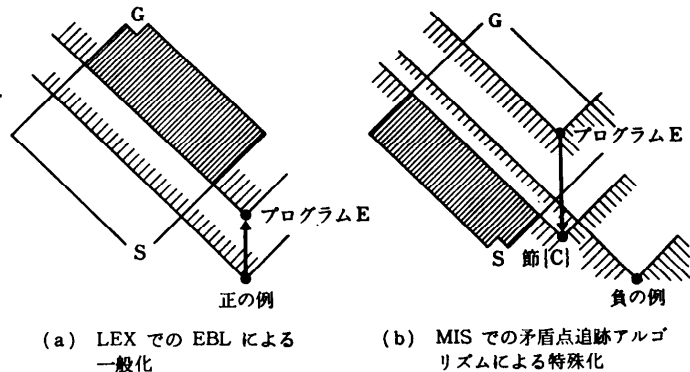
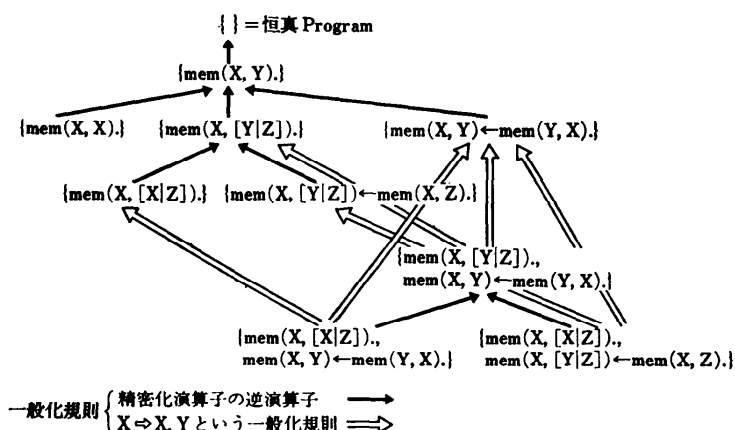


図-2 正の例、負の例の一般性による探索の高速化

間は、考慮する節の全体 (節が一つのみであるプログラムの全体) となっており、精密化グラフ (refinement graph) とよばれる。(節が二つ以上のプログラムも含む) プログラム空間の全体は、精密化演算子 (の逆演算子) と節集合を作るための一般化規則 ($X \Rightarrow X, Y$) によって張られる空間になっている (図-3)。したがって、恒真プログラムからプログラム空間を下方にたどっていくことで、「一般的で簡単なプログラム」から「特殊で、複雑なプログラム」になっていく。これは、MIS が学習結果として好ましいとするプログラムの順序 (選好) を示している。MIS では、プログラムの「一般性」に関する [性質 1], [性質 2] を使ってこのプログラム空間を探索していくことで、すべての例題を正しく解くことのできる「一般的」で「簡単な」(「節数の少ない」) プログラム ($\in G$) を学習する (図-2(b))。

MIS では、楽観的な候補 ($\in G$) を必要とするわけであるから、この探索を効率化するために、 G の更新を行うとき (プログラム P が負の例 e と両立するとき)、誤ったプログラム P の特殊化を行って、候補集合の絞り込みを効率化する。すなわち、 P による e の証明過程で現れる節 $C' = p \leftarrow q (P)$ に対して p, q の真偽を質問することで、効率的に、 p が負の例、 q が正の例となる誤った節 C' を特定する (矛盾点追跡アルゴリズム)。一つの節のみからなる誤ったプログラム $\{C'\}$ は、誤ったプログラム P より特殊なプログラムであるので、この $\{C'\}$ を使えば、プログラム P より効果的に候補集合を絞り込むことができる⁹⁾。

また、MIS での学習では、バージョン空間法、MSC 一般化とは異なり、 G に属する一つのプログラムを保持しているのみである。これは、 G の更新を行うとき

図-3 MIS での $\text{mem}(X, Y)$ のプログラム空間の一部

(プログラム P が負の例と両立するとき), [性質 1], [性質 2] にしたがって, 矛盾点追跡アルゴリズムによって特定される正しくない節 $C(\text{ } \text{ } C')$ のプログラム P からの削除と, 以前に誤りと分かった節を精密化した節の付加を繰り返すことで, 今までのすべての例題を矛盾なく解くことのできる新たなプログラム $P'(\in G')$ を求めることができるためである。

このアルゴリズムを使用することで, 足し算の計算例やソーティングの例を提示することで足し算やソーティングを行う Prolog プログラムを求めることができることが示されている⁹⁾。

【例題の制御による学習の効率化: 負の例, ニアミス, 主体的制御】

プログラム空間を効率的に絞り込むには例題の種類や質を制御することも重要である。たとえば, 正の例だけでなく, 負の例を活用することや, MIS における矛盾点追跡アルゴリズムのように, 効果的な絞り込みができる質問を生成することなどによって例題を主体的に制御することも重要である。

正の例と負の例で 1 か所のみが異なるもの (病気の例でいえば, $A \& B \& \bar{C} \& D$ のときに病気であるが, $A \& B \& \bar{C} \& \bar{D}$ のときは病気でない) をニアミス例というが, このニアミス例も論理積のみによるプログラムを学習する場合, 探索空間を分かりやすく絞り込むのに有効である。すなわち, 探索空間は, ニアミス例で異なっている条件 (病気の例では, D) より特殊なプログラムに絞り込むことができる。Winston は, アーチの概念の学習¹⁰⁾によって, 天井のある 2 本柱はアーチであり, 天井のない 2 本柱はアーチではないなどの

ニアミス例の重要性を示した。

LEX では問題生成器をもっており, プログラムの候補を効率的に絞り込むために必要な問題を生成して問題解決器に解かせるという例題の主体的制御を行っている。このために, ① G の要素 g , S の要素 s に対し, g に該当するが s に該当しない具体的な式 e を問題として提示する, ② 二つの基本演算子のバージョン空間が重なっている場合, その空間中の要素に該当する具体的な式を問題として提示するという二つの戦略をもっている⁷⁾。

【バイアスによる絞り込み】

以上のような一般的な戦略のみでは, 大きなプログラム空間の探索は容易ではない。したがって, 探索空間を実質的に狭くする方法が重要となる。このような観点から学習のバイアス¹¹⁾の重要性が指摘されている。

主として 4 つのバイアスが考えられる。① プログラム空間の絞り込み: プログラムの記述言語に制限を加える。たとえば, 連言のみしか許さなかったり, プログラム空間を張る一般化規則を汎用性の強い規則ではなく, 領域に依存したヒューリスティックや選好に基づいた特殊な規則にする。プログラムの記述言語を制限すると, 与えられた例題を正しく解くプログラムが得られない場合がある。② プログラム間の選好の基準: たとえば, MIS におけるプログラムの簡単さと一般性を合わせたプログラム間の選好がこの例である。Michalski らは, プログラムによる例題の説明度なども含んだ辞書式順序付評価関数 LEF (Lexicographical Evaluation Function) によって, 対象問題に応じ

た選好を明示的に表現している¹²⁾。③プログラム空間の探索法：ヒューリスティックに基づいた探索法である。たとえば、粗い探索をしてから細かく絞り込んでいく高速化手法がある。Michalski らの INDUCE 1.2¹²⁾ は、すべての例題と両立する物体の構造の記述のうちもっとも特殊なもの ($\in S$) を学習するシステムである。そこでは、物体の構造の記述を、物体の属性を表す単項述語 ($large(x)$, $circle(x)$ など) と物体の間の相互関係を示す多項述語 ($on-top(x, y)$, $touch(x, y)$ など) によって行う。プログラム空間の探索には、まず求めるプログラムの多項述語からなる部分のみを決定 (構造のみの空間での探索) し、次に単項述語の部分を決断 (特徴ベクトル空間での探索) する。④背景知識に基づいたプログラム空間の絞り込み：Michalski らの CLUSTER/S¹³⁾ では、背景知識として組み込まれたゴール間依存ネットワーク (Goal Dependency Network; GDN) を使って、プログラムの目的 (ゴール) に応じてプログラムが使用できる述語 (属性) を絞り込む。しかし、このようなバイアスは適切に設定できない場合、学習結果となるプログラムが見つからなかったり、多すぎたりする。

3.3 新述語の導入問題

以上に述べた探索による学習では、プログラムは有限個の既知の述語 (サブルーチン、関数) の組合せによる論理式で定義することができた。ところが、一般には、どんな述語 (サブルーチン、関数) を用意するとプログラム空間を記述できるのかが分からない場合が多く、むしろ、このような述語 (サブルーチン、関数) を求めることが中心となる学習もある。このような学習も、結局はプログラム空間を探索して例題を正しく解くプログラムを求める帰納学習の一つとして捉えることができる。ただし、プログラム空間には (未知の) 無限個の述語が含まれる。

無限個の述語を含むプログラム空間では、初めからすべての述語を考慮することはできない。いつ、どの述語を探索範囲に含めるかが難しい問題となる。この問題は、新述語の導入問題 (new-term problem)¹⁴⁾、あるいは理論名辞の問題とよばれる。

その基本的な戦略は、限られた述語によるプログラム空間で探索してみて、適切なプログラムが見つからなければ、新述語を導入するというものである。導入される新述語は、その意味が既存の述語についての情報から定まるものでなければならない。

この問題は、あらかじめ設定したバイアスが適切で

なかった場合、それを弛める問題と考えることができる。

Langley らの BACON¹⁵⁾ では、陽に新しい述語を導入する規則によって、プログラム空間を定義すると同時に探索の方法も指定している。BACON は、科学データからその法則を求めるプログラムである。そこでの新述語導入ルールとしては、たとえば、系列 x_i と系列 y_i が共に線形に変化するとき、この線形関係の傾きと切片を新しい項目として導入するものなどがある。このような少数のルールによって、ケプラーの法則などいくつかの法則を再発見できた。

LEX のサブシステムである STABB^{11), 16)} では、積分演算子の適用条件を示す記述が既存の一般化束上がない場合、既存の記述の組合せによる新しい記述項目を導入することで新述語の導入問題を扱っている。このために、最小選言手続き (Least Disjunction Procedure) と制約後ろ向き伝播手続きの二つの手続きをもっている。

最小選言手続きは、論理和 (選言) を含んだ記述が必要になった場合、その記述に対応する新しい記述子を一般化束に導入する手続きである。LEX では、論理和 ($\vee$) を含んだプログラムを許さないというバイアスがあるために、この手続きが必要となる。たとえば、 $\int x \sin(x) dx$, $\int x \cos(x) dx$ は部分積分の適用によって計算ができるのに、 $\int x \tan(x) dx$ は部分積分が適用できない。したがって、 $\sin$, $\cos$, $\tan$, $\sec$ の一般化として trig (三角関数) が定義されている場合、部分積分適用の条件 ($x(\sin \vee \cos)(x)$) を一つの項目で記述できない。そこで、 $\sin \vee \cos$ を示す新しい項目を、 trig と $\sin$, $\cos$ の間に導入するように利用者に推薦する。

制約後ろ向き伝播手続きは、前後したように、基本演算子列が成功するための十分条件を求めるものである。この式 ($\int \cos^{\text{odd}}(X) dX$ など) は、既存の一般化束の中の項目として表現されていない場合がありえる。このとき、この項目を新しい項目として一般束に導入して、バイアスを弱くするように利用者に推薦する。この手続きは、既存の一般束で学習を行ったとき、候補集合が空になった場合 (すなわち、一般束が粗すぎて、例と両立する解が見つからなかった場合) に適用される。

4. 演繹学習によるプログラムの生成

前章では、与えられた例題を解くことができる、で

きただけ一般的なプログラムを求める帰納学習について述べた。本章では、演繹学習について述べる。これは、例題が扱っている領域についての基礎知識(公理)は与えられていても、その効率的な利用法(定理)と、その利用のタイミングを知らない場合、基礎知識を利用して効率的に問題を解くプログラム(定理)を求めるものである。帰納学習が論理的能力の学習であったのに対し、これは知識の運用(performance)の学習といえる。このような運用の学習によって、学習前では非常に時間のかかる問題を迅速に解くプログラムを自動生成できる。

4.1 チャンク

認知心理学などの研究により、人間は物ごとを記憶する際に、個々の情報をそれぞれ個別に憶えるのではなく、チャンク(chunk)とよばれる情報のかたまりを構造的に記憶していると考えられている¹⁷⁾。問題解決の時点においても、“この状況ではこれこれの手順で解いていけばよい”という形のチャンクとして記憶された知識を活用することで問題を解くことが、しばしば観察される。

機械学習においても、同様に、すでに解いたことのある問題に関して、そのパラメータと結果、解決の手順などをチャンクとしてまとめることで、より効率的な問題解決プログラムが得られる。

Korf は、チャンク(マクロオペレータ)を階層的に導入して問題分割を行う(抽象化階層: abstraction hierarchy)ことで問題解決の計算量が指数から線形におとせることを指摘している¹⁸⁾。

チャンク化の形態は、学習されるプログラムがどのような計算原理に基づくかによって異なる。

① マクロルール化: 一階述語論理、プロダクションシステムなど、ルールによる推論に基づくプログラムにおけるチャンク。いくつかのルールの条件部と結論部を合成し、新しいルール(マクロルール)を作る。たとえば、 $R1: A \leftarrow B \& C$, $R2: B \leftarrow D \& E$ が問題解決に使われたとき、二つのルールを合成し、単一のマクロルール $\text{macro}R: A \leftarrow D \& E \& C$ のチャンクにする。

② マクロオペレータ化: 三角表による目的手段解析(MEA: Means-End Analysis)に基づくプログラムでのチャンク。あるゴールを達成するために使用した特定のオペレータ列 $OP_1, OP_2, \dots, OP_n$ を単一化したマクロオペレータ $\text{macro}OP = OP_1, OP_2, \dots, OP_n$ のチャンクにする。

③ テーブル参照法: 特定の判別関数に基づくプログラムでのチャンク。たとえばメモ関数は、判別関数 $f(x)$ の他に、例題を解いた際の入力パラメータ a と、その結果 $f(a)$ との対 $\langle a, f(a) \rangle$ の表を補助テーブルとしてもつ関数である。通常は補助テーブルを参照し、入力パラメータに対応する結果があるかを調べる。結果があるときはそれを関数値とし、ないときのみ判別関数の計算が行われる。計算結果は、以降の処理のために、パラメータ値とともに補助テーブルに格納される。

4.2 問題解決の総コスト

学習結果となるプログラムがいかに効率的であっても、それを学習するために膨大な計算量が必要であるならば、多少効率が悪くても簡単に得られるプログラムを用いて解を求めたほうが良い場合もある。したがって、学習の効率化は、例題列から効率的な問題解決プログラムを得るための学習コストと、学習されたプログラムによって特定の問題を解くための問題解決コストの和からなる総コストで測ることが望ましい(総コスト = 学習コスト + 問題解決コスト)。したがって、効率の向上とは、この総コストをできるだけ小さくすることである。

一般に、効率的なプログラムを得るためのテクニックには、サブルーチン展開や冗長コードの除去などの対象問題の性質といった問題全体に関する情報を必要としないローカルな手法と、部分問題への分割やアルゴリズム全体の変換などの問題全体に関する情報を必要とするものがある。前者は、学習コストは小さくて済む一方、問題解決コストはあまり小さくならないことが多い。これに対し後者は、問題解決コストは低く抑えられるが、学習コストは大きい。

マクロルール化などのチャンクを得る操作自身は、コストを要しないローカルな効率化手法である。コストを要するのは、チャンクとしてまとめる一群を見つける操作であるから、どのような一群をチャンクとしてまとめるかが問題となる。

4.3 チャンクを求める学習

問題空間のグローバルな情報を利用すれば、たとえば、縦続分解可能(serially decomposable)^{19), 20)} な問題の場合、問題解決の総コストを抑えた学習が行える。

しかしながら、このような性質の良いクラスに属する問題は限られており、一般の問題について、学習コストを抑えながら効率の良いプログラムを得る方法は重

要である。このような方法の一つとして、具体的な例題の解法から、その解法が一般に適用できる条件と結果の形を求め記憶しておくことで、例題の類題に対しての問題解決を効率化する説明に基づく学習 (Explanation-Based Learning: EBL) がある。

〔説明に基づく学習〕

演繹学習では、一般的に対象問題 G を原理的に解くことのできるプログラム P が背景知識として与えられている。説明に基づく学習 (EBL) は、まず、このプログラム P を用いて、対象問題 G についての具体的な例題 e を解いてみる (説明フェーズ)。次に、その具体的な例題の解決手順 (トレース) を基に、その類題を効率的に解くことのできる一般的なプログラム Q を求める (一般化フェーズ) というステップから成る。したがって、得られるプログラム Q は、背景知識であるプログラム P より特殊ではある ($Q \models P$) が、効率のよいものとなっている。

〔説明に基づく一般化〕

代表的な説明に基づく学習として、説明に基づく一般化 (Explanation-Based Generalization: EBG)^{20,21)} がある。説明に基づく一般化では、対象問題 G を目標概念 (Goal Concept)、具体的な例題 e を訓練例 (Training Example) とよぶ。プログラム P は、扱う問題領域に固有の領域知識 (Domain Theory) とよばれる知識と、目標概念の記述によって与えられる。学習されるプログラム Q の効率性についての判定基準を操作性規範 (Operational Criteria) とよび、 Q は目標 G の記

述で、操作性規範を満たすものである。操作性規範としては、一般に、目標概念が計算コストのかからない述語で記述されるという条件や、直接観測可能な述語で記述されるという条件が採用される。

LEX で扱われた問題を例にとりあげる⁷⁾。目標概念は、「ある積分オペレータ OP をどの形の式 $E0$ に適用すると積分に成功するか」を示す述語「正例 ($OP, E0$)」である。図-4 に目標概念の記述と領域知識を示す。この二つによって、この問題を (原理的に) 解くことのできるプログラム P が与えられている (ここでのルール及び事実の記述のための構文は、Prolog に準ずる)。このとき、目標概念が、Prolog の組み込み述語と定数 (X)、積分を含まない (X) によって記述されるという操作性基準を満たすルールを求めることを考える。まず、説明フェーズでは、正の例 $\int 5x^2 dx$ を訓練例として、その解き方をプログラム P により求める。このルールの適用の仕方は、図-5(a) に示すような証明木となる (この証明木を、EBL では説明木と言う)。組み込みの問題解決器で解くことができない場合、教師 (外界) から解き方を教えてもらっても良い。

一般化フェーズでは、正例 ($OP, E0$) (ただし、 $OP, E0$ は自由変数) に、図-5(a) で得られた説明木にしたがってルールを適用する。この結果、図-5(b) のような一般化された説明木が得られる。これをチャンク化することで、学習結果として次のルールが得られる。

正例 ($op\ 2$, 積分 ($K * X^N, X$)): -

- 〔目標概念〕《正例 (演算子, 式): 式に演算子を適用すると、積分可能になる。》
 正例 ($OP, E0$): - 適用 ($OP, E0, E1$), 可解 ($E1$).①
- 〔領域知識〕《適用 (演算子, 式 1, 式 2): 式 1 への演算式の適用結果は式 2》
 適用 ($op\ 1, X * Y, X1 * Y1$): - 適用 ($OPX, X, X1$), 適用 ($OPY, Y, Y1$). 《積の公式》②
 適用 ($op\ 2$, 積分 ($K * F, G$), $K * \text{積分} (F, G)$): - 定数 (K). 《定数係数の積分公式》③
 適用 ($op\ 3$, 積分 (X^N, X), $1/M * X^{N+1}$): - M is $N+1$. 《 n 乗の積分公式》④
 適用 (nop, X, Y).⑤
 《可解 (式): 式が積分可能 (適当な演算子列の適用で積分を含まない式が得られる)》
 可解 ($E0$): - 終了 ($E0$).⑥
 可解 ($E0$): - 適用 ($OP, E0, E1$), 可解 ($E1$).⑦
 《終了 (式): 積分を終了した式》
 終了 ($X * Y$): - 終了 (X), 終了 (Y).⑧
 終了 ($X + Y$): - 終了 (X), 終了 (Y).⑨
 終了 ($X \wedge Y$): - 終了 (X), 終了 (Y).⑩
 終了 (X / Y): - 終了 (X), 終了 (Y).⑪
 終了 (X): - 定数 (X).⑫
 終了 (X): - 積分を含まない (X).⑬
 《積分を含まない: 積分記号が含まれない式》
 積分を含まない (X): - <詳細省略>
- 〔操作性規範〕《目標概念が組み込みの述語 (+, *, など) と, 「定数」, 「積分が含まれない」で記述される》
 operational (P): - P =定数 (X); P =積分を含まない (X); built.in (P).

図-4 不定積分問題を解くプログラム P

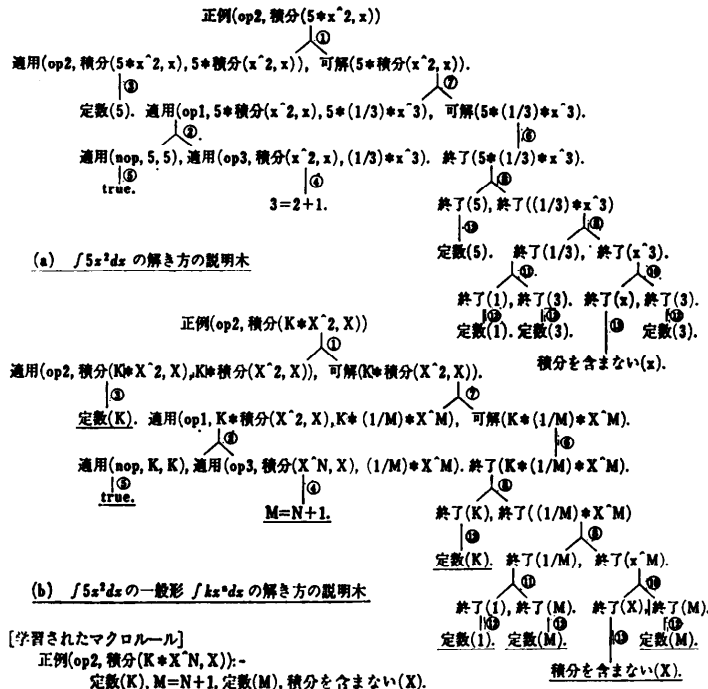


図-5 プログラム P からの説明に基づく学習

定数(K), $M = N + 1$, 定数(M), 積分を含まない(X).

すなわち, 一般に $\int kx^2 dx$ を求めるときには op2 を適用すればよい. これは op2 適用のための十分条件を求めたものであり, 例題 ($\int 5x^2 dx$) から類題 ($\int kx^2 dx$) を解くための効率の規則を求めたことになる.

LEX のサブシステム STABB^{11), 16)} では, この一般化フェーズを, 制約後ろ向き伝播手続きによって行っている. この手続きは, 説明フェーズで求められた積分オペレータ列 $OP_1, \dots, OP_n$ が積分に成功する条件を求めるもので,

原像 $(OP, F) = \{x | x \in OP \text{ の適用領域} \ \& \ OP(x) \in F\}$ を用いて, オペレータ OP_n から OP_1 に向かって, オペレータを適用したときと逆方向にその原像を求める操作を繰り返す. すなわち,

[オペレータ列 $OP_1, \dots, OP_n$ が成功する式の集合]
= 原像 ($OP_n, \dots, OP_1$, 積分を含まない式の全体 NI)
= 原像 (OP_1 , 原像 ($OP_2, \dots$, 原像 (OP_n , NI))).

具体例の証明木と, それを一般化した証明木の構造が同一であることに着目すると, Prolog のメタイン

タプリタを拡張することで, 説明フェーズと一般化フェーズを同時に行うことができる. そのプログラムを次に示す²²⁾.

```
ebg ((Goal1, Goal2), (GenGoal1, GenGoal2),
    (Leaves1, Leaves2)) :-
    !, ebg (Goal1, GenGoal1, Leaves1),
    ebg (Goal2, GenGoal2, Leaves2).
ebg (Leaf, GenLeaf, GenLeaf) :-
    operational (Leaf), !, call (Leaf).
ebg (Goal, GenGoal, Leaves) :-
    clause (GenGoal, GenClause),
    copy ((GenGoal :- GenClause), (Goal :- Clause)),
    ebg (Clause, GenClause, Leaves).
copy (Old, New) :- assert ('$marker'(Old)),
    retract ('$marker'(New0)),
    New = New0.
```

ここで述語 ebg (<訓練例>, <目標概念>, <十分条件>) は, 第一引数として訓練例を, 第二引数として目標概念を与えると, 第三引数に一般化された説明木の葉 (Leaf) の並びが得られる. 説明木の葉となるのは, その述語が操作性基準を満たすかを判定する述語

operational を満たしたものである。〈目標概念〉:-〈十分条件〉が、学習の結果得られる操作性基準を満たすルールである。たとえば、図-4 の例を適用すると、

? :- ebg(正例(op 2, 積分($5 * x^2$, x)),

正例(OP, E0), Leaves)).

から、図-5(b)の説明木の葉(下線部)を集めた、

正例(op 2, 積分($K * X^N$, X)) :-

定数(K), (true, $M = N + 1$), 定数(K),

(定数(1), 定数(M)),

積分を含まない(X), 定数(M).

が学習結果として得られる。

この EBG プログラムは、Prolog による部分計算のプログラムときわめて類似している。一般に、EBL は例題によってガイドされた部分計算器と捉えることができる²³⁾。

EBL の考え方に基づいていくつかの実験システムが作られている。Winston の CUP²⁴⁾ は、機能的記述を目標概念としてその構造的記述を得る。また、LEX^{21), 20)}, PRODIGY^{25), 26)} などは、オペレータの適用の成功や失敗を目標概念として、メタルールの形で領域レベルの制御知識を学習する。

[説明に基づく一般化における操作性]

プログラム空間の立場からみると、EBL は、元となるプログラム P (= 領域知識 + 目的概念の記述) より特殊なプログラムで、操作性規範を満たす極大のプログラム Q を求めるものである。ただし、操作性規範 operational には、以下の単調性が仮定される²⁷⁾。

P ≧ Q のとき、operational(P) → operational(Q)。

実用的な問題を扱う場合、目標概念 G を原理的に解くことのできる領域知識 D では、実際に説明構造を生成するには時間がかかるので、以前の訓練例から学習されたルールなどの計算コストの低いマクロルールを使用して説明構造が作られる。こうして得られた説明構造の一般化によって得られるルールは、学習済みのルールを使わない場合よりも特殊なものとなる可能性がある。Braverman らは、これを防ぐために、適切に一般化されたルールを得る手法を提案している²⁸⁾。

EBG では、単一の例題を元にその種類を効率的に解くルールを学習した。したがって、この学習されたルールは類題以外には適用できない。より多くの問題に対しても適用できるようにするには、複数の例題から学習された複数のルールによって問題を解くことが考えられる。その結果、さまざまな種類の類題に対応できるようになる。しかしながら、学習が進みルール

数が増えるにつれて、システムの効率が向上しないことが報告されている^{29), 30)}。これは、EBG で与えられる操作性規範は、ルールがどのように使われるかを明確に捉えていないことによる。Keller は、操作性規範を学習後のシステムでの利用可能性(usability)と効用(utility: 学習によって効率が向上すること)によって定義しなおしている³¹⁾。Minton は、効用の高いルールを学習結果として維持していくメカニズムを採用した場合の効果について、いくつかの応用問題についてその定量的評価を示している³²⁾。

5. ま と め

1980年代前半までの機械学習の研究は、帰納学習についての研究が中心であった。これは、可能なプログラムの集合の中から、求めるプログラムを効率よく探索する問題として捉えることができ、ポイントは、実質的に無限の大きさをもつ探索空間をどう与えるかということと、これをいかに効率的に探索するかということであった。前者は、一般化規則などによってプログラム空間を張り、必要に応じて新しい述語を付加してプログラム空間を拡張していくことで行う。後者は、プログラム空間をあらかじめ小さく抑えたり、プログラム間の選好の基準をはっきりさせたり、プログラム空間中の探索法の選択といった学習に対するバイアスの制御によって行う。このバイアスが適切に設定しにくい問題に対しては、効率的な学習ができない。たとえば、例題の真偽に誤りが含まれる場合などには、適切なバイアスが設定しにくく、効率的な学習が行いにくい。また、帰納学習では、一般性の高いプログラムほど望ましいプログラムとされることが多い。したがって、論理的根拠がなく、類似性のみに基づいた一般化がなされ過ぎる場合があり、得られたプログラムの正しさは保証されず、実際にこれによって得られたプログラムを使って問題を解くことが危険である場合もある。

そこで、1980年代中ごろからの機械学習の研究は、与えられた問題を原理的に解けるだけの知識が与えられた上で、これらの知識の効率的な活用法を学習する演繹学習が注目されるようになってきた。

特に、まずあらかじめ与えられた知識(プログラム)を使って与えられた例題を解き、次にその解き方を調べることで、同じような解き方を適用するための(十分)条件を見つけ出す、説明に基づく学習(EBL)が注目されている。ただし、原理的に問題を解けるだけ

の、誤りを含まない知識（領域知識）があらかじめ与えられていることが大前提である。したがって、不完全な、あるいは仮説を含んだ領域知識に基づいた学習をいかに行うかが問題となる。

このように帰納学習、もしくは、演繹学習の一方では、論理能力を高めながら効率の良いプログラムを得ることは難しい。類似の問題や関連した領域での解法やプログラムを現在の問題にも適用できるようにする類推学習^{33), 34)}などを含め、今後は両者をいかに融合するかが重要な点である。

参 考 文 献

- 1) Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.): *Machine Learning: An Artificial Approach*, Tioga (1980). (邦訳: 知識獲得と学習シリーズ 1～3 巻, 共立出版).
- 2) Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.): *Machine Learning: An Artificial Intelligence Approach*, Vol. 2 (1983). (邦訳: 知識獲得と学習シリーズ 4～8 巻, 共立出版).
- 3) Michalski, R. S.: *Understanding the Nature of Learning: Issues and Research Directions*, in 2), ch. 1.
- 4) Mitchell, T. M.: *Version Spaces: A Candidate Elimination Approach to Rule Learning*, Proc. of IJCAI, pp. 305-310 (1977).
- 5) Michalski, R. S.: *A Theory and Methodology of Inductive Learning*, in 1), (1980).
- 6) Vere, S. A.: *Induction of Concept in the Predicate Calculus*, Proc. of IJCAI, pp. 281-287 (1975).
- 7) Mitchell, T. M., Utgoff, P. E. and Banerji, R. B.: *Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics*, in 1), ch. 6 (1980).
- 8) Shapiro, E. Y.: *Algorithmic Program Debugging*, MIT PRESS (1983).
- 9) 有川節夫, 石坂裕毅: 帰納推論による自動プログラミング, 情報処理, Vol. 28, No. 10 (1987).
- 10) Winston, P. H.: *Learning Structured Description from Examples*, The Psychology of Computer Vision, Winston, P. H. (Ed.), McGraw Hill, ch. 5 (1975).
- 11) Utgoff, P.: *Machine Learning of Inductive Bias*, Kluwer Academic Publisher (1986).
- 12) Dietterich, T. G. and Michalski, R. S.: *Inductive Learning of Structural Description: Evaluation Criteria and Comparative Review of Selected Methods*, Artificial Intelligence, Vol. 16, No. 3, pp. 257-94 (1981).
- 13) Stepp, R. E. and Michalski, R. S.: *Conceptual Clustering: Inventing Goal-Oriented Classification of Structured Objects*, in 2), ch. 17 (1983).
- 14) Cohen, P. R. and Feigenbaum, E. A.: *Learning from Examples*, in the Handbook of Artificial Intelligence, Vol. 3, ch. 14 D 1, Morgan Kaufmann (1982).
- 15) Langley, P. et al.: *The Search for Regularity: Four Aspects of Scientific Discovery*, in 2), ch. 16 (1983).
- 16) Utgoff, P.: *A Shift of Bias for Inductive Concept Learning*, in 2), ch. 5 (1983).
- 17) Rosenbloom, P. S. and Newell, A.: *The Chunking of Goal Hierarchies: A Generalized Model of Practice*, in 2).
- 18) Korf, R. E.: *Learning to Solve Problems by Searching for Macro-Operators*, Morgan Kaufmann (1985).
- 19) Korf, R. E.: *Planning as Search: A Quantitative Approach*, Artificial Intelligence 33, pp. 65-88 (1987).
- 20) Mitchell, T. M., Keller, R. M. and Kedar-Cabelli, S. T.: *Explanation-Based Generalization: A Unifying View*, Machine Learning 1, pp. 47-80 (1986).
- 21) DeJong, G. F. and Mooney, R.: *Explanation-Based Learning: An Alternative View*, Machine Learning 1, 2 (1986).
- 22) Keder-Cabelli, S. T. and McCarty, L. T.: *Explanation-Based Generalization as Resolution Theorem Proving*, Proc. of the 4th Int. Workshop on Machine Learning, pp. 383-389, Morgan Kaufmann (1987).
- 23) Harmelen, F. and Bundy, A.: *Explanation-Based Generalization=Partial Evaluation*, Artificial Intelligence 36, pp. 401-412 (1988).
- 24) Winston, P. H., Binford, T. O., Katz, B. and Lowry, M.: *Learning Physical Descriptions from Functional Definitions, Examples and Precedents*, Proc. AAAI-83, pp. 433-439 (1983).
- 25) Minton, S. et al.: *Acquiring Effective Search Control Rules: Explanation-Based Learning in the PRODIGY System*, Int. Workshop on Machine Learning (1987).
- 26) Minton, S. and Carbonell, J.: *Strategies for Learning Search Control Rules: An Explanation-based Approach*, Proc. IJCAI-87, pp. 228-235 (1987).
- 27) Braverman, M. S. and Russell, S. J.: *IMEX: Overcoming Intractability in Explanation Based Learning*, Proc. AAAI-88, pp. 575-579 (1988).
- 28) Braverman, M. S. and Russell, S. J.: *Boundaries of Operability*, 5th Int. National Conference on Machine Learning (1988).
- 29) Minton, S.: *Selectively Generalizing Plans for Problem Solving*, Proc. IJCAI-85 (1985).
- 30) Tambe, M. and Newell, A.: *Some Chunks*

- are Expensive, 5th Int. National Conference on Machine Learning (1988).
- 31) Keller, R. M.: Defining Operationality for Explanation-Based Learning, Artificial Intelligence 35, pp. 227-242 (1988).
- 32) Minton, S.: Quantitative Results Concerning the Utility of Explanation-Based Learning, Proc. AAAI-88, pp. 564-569 (1988).
- 33) Carbonell, J. C.: Learning by Analogy: A Formulation and Generalizing Plans from Past Experience, in 1), ch. 5 (1983).
- 34) Davis, T. R. and Russell, S. J.: Logical Approach to Reasoning by Analogy, IJCAI-87 (1987).

(昭和 63 年 12 月 9 日受付)
