問題解決における戦略知識の学習

山 田 誠 二

平成元年２月

大阪大学基礎工学研究科

大阪大学基礎工学研究科　博士論文

問題解決における戦略知識の学習

山田誠二

平成元年 2月

要 旨

本論文は，問題解決において戦略知識を機械学習により獲得する方法論について書かれたものである．この研究はPiLプロジェクトと呼ばれ，主に説明に基づく一般化：EBGとマクロオペレータの選択的学習を基盤にして理論的展開がなされ，その実験的検証のためのフレームワークとしてPiL，PiL2という2つの学習システムが構築された．PiLプロジェクトにおいて提案された機械学習に関するこれまでにない新しい手法は，以下の2点に集約される．

1）操作可能な解決可能概念（SOLVABLE概念）を定義し，それを用いることによってEBGの過小一般化（under-generalization）を改善する「直接解決可能性に基づく一般化：DSBG」.

2）完全因果性という新しいヒューリスティックに基づき，多数ある候補から有効なマクロオペレータだけを選択的に学習する手法.

以下，簡略に上記2点について説明していく．

1）解決可能概念とは，最終的に解決できる問題状態の集合である．これはMitchellらによって最初に提案されたが，ある問題状態が解決可能概念に含まれるかどうかの検証が実際に問題を解いていくのと等価であったために操作可能（operational）な概念記述ではなかった．その後，Kellerによって解決可能概念の操作可能化が成されたが，その手法は非常に単純で厳密な概念記述は得られなかった．そこで，本論文では完全因果性によって得られるワンステップで問題解決可能なマクロオペレータを用いて，操作可能な解決可能概念をより正確に定義した．その解決可能概念を用いてマクロオペレータの説明木を構成することにより，これまでよりもEBG

の説明木の深さが減り，より十分な一般化が可能になる．その結果，PiLに与える必要のある訓練例の数が大幅に減少し，学習効率が向上することが，各種方程式において実験的に検証された．

2) マクロオペレータとは，入力として与えられた複数のオペレータで構成されるオペレータシーケンスを一つのオペレータに合成したものである．このマクロオペレータの適用により，複数ステップの適用がワンステップで行なわれ，問題解決の効率が向上する．しかし，一般に過去の解法過程から考えられるマクロオペレータの候補は非常にたくさんあり，その中から選択的に学習しないとかえって学習なしシステムよりも問題解決の効率が低下してしまう．このマクロの選択についていままでいくつかの手法が提案されてきたが，そのいずれにも問題点があった．本論文では，完全因果性という問題領域から独立なマクロ選択のヒューリスティックを用いて，有効かつ少数のマクロオペレータだけを生成する手法を提案し，各種方程式及びロボットの行動計画の分野でその有効性を確認した．

以上の2点はいずれも単なるアプリケーションではなく，機械学習の方法論における根本的提案であり，それぞれ実験的に可能な範囲で検証されたと考える．

＜ 目 次 ＞

第1章　序論

機械学習（Machine Learning）は人工知能（Artificial Intelligence）における本質的課題である．知能あるいは知性の定義付けには，必ず「自ら学習する能力をもっていること」という条件が付随するといって過言ではない．つまり，学習能力のないシステムは，本質的には人工知能システムとは言えないのである．このような観点から，長年にわたり機械学習に関する研究が続けられてきた．これまでの機械学習の研究は，非常に多数の訓練例を与えられ，それらの共通要素を抽出することにより一般化し，学習する「帰納的学習」がその主流を占めていた．

しかし，2章で詳述するように，これまでの帰納的学習は訓練例をどの概念にまで一般化するかという一般化のレベルの設定や訓練例の共通部分検出のヒューリスティックの根拠が明確に示されておらず，学習システムとして不透明な印象を受ける．これは帰納的学習が論理的に説明不可能な知識を対象としている場合が多々あるために不可避的なことなのではあるが，説明可能な知識を対象としている場合は学習の方法論として明確さに欠ける．これに対して，訓練例がなぜ学習すべき概念に含まれるのかを領域固有の知識を用いて明確に説明し，その説明が成り立つ範囲で一般化を行なう「説明に基づく学習：EBL（Explanation－Based Learning）」が提案された．EBLにおいては，論理的に説明できる知識しか学習できないが，入力として与えられるべき知識と学習により得られる知識が一般性をもって明確に定義されている．そのため，説明可能な知識の学習には帰納的学習よりも見通しがよい．さらに一つの訓練例からでもそれを説明することにより一般化可能なので学習効率もよく工

学的価値が高い.

一方，入力として与えられたオペレータの特定のシーケンスを一つに合成したマクロオペレータを学習し，効率を向上させるマクロオペレータの研究もSTRIPS [Fikes 72] 以来行なわれてきた．マクロオペレータの研究が始まった当初は，考えられるすべてのマクロをとにかく蓄積して行き，そこから随時必要なものを抽出することに重点が置かた [Fikes 72]．しかし，その後過去の解法過程をすべてマクロオペレータとして残しておけば蓄積されるマクロは爆発的に増加し，その中からいま必要なマクロを選択するのに非常にコストがかかることがわかってきた．さらにその結果，有限個のそれもかなり少数の問題を学習させた場合でも，マクロの急増によりマクロ学習システムの問題解決の効率が学習無しシステムよりも低下してしまうという興味深い現象が報告されている [Minton 85]．よって，現在ではマクロオペレータ学習の研究の最大課題は数多くあるマクロオペレータの候補から，どのように役に立つ少数のマクロを選択的に学習するかということになっている.

本論文は，問題解決における探索制御のヒューリスティックとマクロオペレータの学習を目指した研究である，PiLプロジェクトについて述べたものである．PiLプロジェクトは，1）戦略知識学習システム：PiL（本論文第3章），2）直接解決可能性に基づく一般化：DSBG（第4章），完全因果性によるマクロオペレータの選択的学習（第5章）という3つの研究から成る．1）のPiLシステムは方程式の分野におけるEBGのアプリケーションとマクロオペレータ学習の新しい方法論の提案であり，実際に各種方程式の解法学習においてその有効性を確認した．2）のDSBGは，新しい操作可能なSOLVABLE概念の定義付けを提案し，それを用いた

一般化手法：DSBGをPiLシステムに適用して，従来の最終状態からのEBGによる一般化よりもさらに十分な一般化が可能であることを示す．さらに，1）での方程式の分野で，解法を教示しなければならない訓練例がDSBGにより大幅に減少することを確認する．さらに，3）では1）で提案され方程式の分野で有効であった完全因果性によるマクロオペレータの選択的学習の手法が，STRIPSのロボットの行動計画においても有効であることを，汎用マクロオペレータ学習システム：PiL2を用いて検証する．

第2章　これまでの機械学習研究

PiLプロジェクトの本論にはいる前に，本章ではこれまで行なわれてきた機械学習に関する研究を傍観し，この分野の研究の流れを把握する [Watanabe 88] [Numao 88]．一般的に機械学習の研究は，1）帰納的学習，2）演繹的学習，3）発見的学習，4）類推による学習，に分類される．まず，それぞれについての概略を述べる．

1）帰納的学習

帰納的学習は例題からの学習が一般的である．学習すべき概念の要素である訓練例を「正の訓練例」，その概念の要素でない訓練例を「負の訓練例」と呼ぶ．帰納的学習は最近では「類似に基づく学習（Similarity-Based Learning）」と呼ばれるように，正の訓練例からその構文的（syntactic）な共通部分を検出することにより一般化を行う [Michalski 83]．そのとき，どの概念にまで一般化するかを決定するために事前に概念ハイアラーキが与えられることが多い．概念ハイアラーキとは，いま学習が行なわれる領域における概念の一般／特殊あるいは全体／部分などの包含関係を表わした階層である．帰納的学習システムは明示的（explicit），暗示的（implicit）に関わらずこの概念ハイアラーキを与えられている．しかし，この概念ハイアラーキをどんな場合でも正しい一般化を導くように，事前に設計することは非常に困難なこと，また実際の学習過程で状況に応じて概念ハイアラーキを動的に変化させることが難しいこと，などが帰納的学習の問題点である．

また，記号レベルでの帰納的学習とは異なった学習手法として，近年再

び活発に研究され始めたニューラルネットワークがある．「人間は知っていると思っていること以上を知っている」と言ったのはM．ポランニー[Polanyi 66]であるが，ニューラルネットによる学習は，人間のもっている知識のうち対象化できない知識，いわゆる無意識に用いている知識を学習しようというもので，記号レベルでの学習と同じく訓練例により学習が行われる．ニューラルネットの最も一般的なモデルは，McCulloch and Pittsの多入力1出力のニューロンが数多くリンクされたモデル[McCulloch 43]である．ニューラルネットワークの学習は，このニューロン間のリンクの重みを変化させることによって実現される．教師により正の訓練例と負の訓練例を与えられ，それらの例に対して正しい結果がでるようにニューロン間のリンクの重みを変化させる方法（バックプロパゲーション法）[Rumelhart 86]やホップフィールドモデル[Hopfield 85]など種々のモデル，学習方式が提案されている．

2）演繹的学習

前述の帰納的学習の問題を克服するものとして，ここ数年機械学習の分野でのトピックスとなっている学習方式である．具体的に現在行われている演繹的学習の研究のほとんどは，「説明に基づく学習：EBL (Explanation-Based Learning)」に関するものである．EBLは与えられた訓練例が，なぜ学習されるべき目標概念に含まれるのかを領域知識(Domain Theory）と呼ばれる現在対象としている問題分野に固有の知識を用いて説明する．この説明の過程が「説明木」とよばれるもので，この説明木を各領域知識を満たす範囲で一般化していく．このため帰納的一般化とは対照的に一つの訓練例からでも一般化が可能であり，非常に学習

の効率がよい．また，帰納的学習では一般化が静的な概念ハイアラーキによって誘導されるので，常に根拠のある正しい一般化がなされる保証はない．それに対してEBLでは説明木が一般化の根拠を構成しており，その一般化の正当性を保証している．

3）発見的学習

これまで述べてきた学習方式は学習すべき概念の骨格となる記述を与えられていた．発見的学習は与えられた目標概念の記述だけでなく，そこから派生する別の概念記述を発見的に学習していく．インプリメントされた発見的学習システムとしては，レナートのAM，Eurisko [Lenat 83] が非常に有名である．AMは最初に集合論に関する人間の数学者の数学的美意識や価値観（例えば，「定義域と値域の一致する関数はおもしろい」など）をヒューリスティックとして与えられ，それに基づいて自然数，素数，四則演算などの概念を自動的に発見していく．AMで発見された概念の一部は意味のある重要な概念であり，その中には人間の数学者の見落としていた概念も含まれていたといわれている．EuriskoはAMをさらに改良したもので，ヒューリスティック自身をヒューリスティックから発見的に学習しようというものであった．このシステムは戦艦ゲームやVLSI設計の分野において成果を上げた．

4）類推による学習

「類推」とは二つの対象をある抽象レベルで対応付けることである．類推による学習とは過去に得られた問題解決過程などを新しい対象に対応付けることにより利用しようというもので，その先駆的研究は，Winston

によるマクベスのプロットの類推学習［Winston 80, 83］である．これはシェークスピアのマクベスにおける各登場人物の性格や人間関係からどのような事件が起こったかを一般化して記憶し，次によく似た物語が与えられたときに起こり得る事件を想定することが可能である．なお，類推に関しては最近，原口ら［Haraguchi 86］によって1階述語論理（ホーン節）の範囲で定式化がされている．

以上，機械学習の各分野について概観してきた．次節以降では本研究と関連のある分野について学習システムの解説を中心にさらに詳しく述べていくことにする．具体的には，PiLプロジェクトのテーマである問題解決における効率の向上を目指した学習に関連した研究について触れていく．

2. 1 類似に基づく学習（Similarity-Based Learning）
-LEXシステムを中心に-

帰納的学習あるいは類似に基づく学習全般については［Michalski 83］等を参考にしてもらうとして，ここでは問題解決の分野で帰納的学習により探索制御ヒューリスティックを獲得するシステムとして高名なMitchellらのLEXシステム［Mitchell 81, 83a］について詳述していく．

2. 1. 1 LEXシステムの概要

LEXは不定積分の分野において入力として与えられる基本オペレータをどのように適用すれば効率よく問題が解けるかという問題解決の制御

用ヒューリスティックを学習するシステムである．LEXは，初期状態として不定積分のオペレータをもち，ヒューリスティックをまったくもっていない状態から初めて，徐々にそのオペレータを適用すべき状況を学習していく．このオペレータが適用されるべき状況（問題状態）が，LEXが学習する目標概念である．それぞれのオペレータの前提条件は，最初与えられたときは最も一般的なものなので非常に数多くの無駄な適用がなされるが，それが学習によってより限定された適用範囲に洗練されていく．その結果，無駄な適用がなくなり問題解決の効率が向上する．この洗練された適用範囲がLEXが学習すべき概念記述であり，条件部が洗練されたオペレータが制御用ヒューリスティックである．ここでは，練習問題を解く過程から得られたオペレータの適用例（これが正の訓練例）を一般化することにより条件部の洗練が行われる．

Fig.2.1にLEXに与えられるオペレータを表わす．また，下に学習されるヒューリスティックの具体例を示す．

$\int f(x)\ \mathrm{transc}(x)\ dx \rightarrow$ Apply OP2（部分積分）

$[u = f(x),\ dv = \mathrm{transc}(x)\ dx]$

これは，「もし問題中にxとxの超越関数の積の積分があれば，uとdv

OP1 : $\int rf(x)\ dx \rightarrow r \int f(x)\ dx$
OP2 : $f\ u\ dv \rightarrow uv - \int v\ du$ （部分積分）
（内部では$\int f1(x)\ f2(x)\ dx$と表現されている
$f1(x) = u,\ f2(x)\ dx = v$）
OP3 : $1f(x) \rightarrow f(x)$
OP4 : $\int \{f1(x) + f2(x)\}\ dx \rightarrow \int f1(x)\ dx + Ff2(x)\ dx$
OP5 : $\int \sin(x)\ dx \rightarrow -\cos(x) + C$
OP6 : $\int \cos(x)\ dx \rightarrow \sin(x) + C$
OP7 : $\int x^r dx \rightarrow x^{r+1}/(r+1) + C$

Fig.2.1 LEXの積分オペレータの例

を［］内のようにおいて，部分積分を試みよ」というヒューリスティックである．このヒューリスティックはFig.2.1のOP2を洗練したものであるが，Fig.2.1では条件部のf2（x）が任意の関数であるのに対し，上記のヒューリスティックではf2（x）はtransc（x）に特殊化され，洗練されたのがわかる．

2．1．2　LEXの学習方式

LEXはFig.2.2に示すような4つのモジュールから構成されている．この図をもとにLEXの学習を説明していく．まず，問題生成部が現在獲得されているヒューリスティックをより正確なものにするために有効と考えられる練習問題：∫ 3xcons（x）dxを生成し，問題解決部へ渡す．ただし，Fig.2.2の場合はOP2についてのヒューリスティックはまったく無い

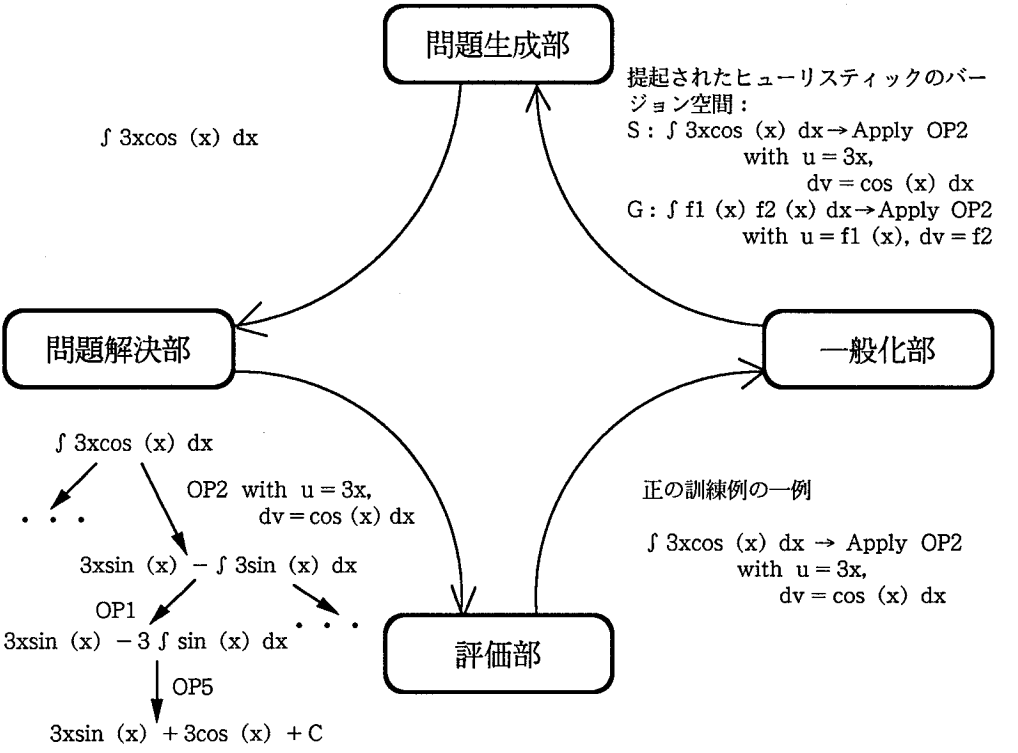


Fig.2.2　LEXの学習サイクル

状態とする.

それを問題解決部が解く. ここでの探索戦略は, 前方探索で問題に適用できるヒューリスティックがあれば優先的に適用するが, 無ければ適用できるオペレータを逐次適用していく. そして, 一定の演算時間とメモリ容量の制限内で解ければその解決過程が批評部にわたされる. もし, 制限内に解けない場合は, その問題は捨てられて次の問題に移る. この場合は {OP2, OP1, OP5} というオペレータシーケンスの適用で解決され, その解決過程が評価部へ渡される.

解決過程を受け取った評価部は, そこでのオペレータの個々の適用例が正の訓練例であるのか, 負の訓練例であるのかを評価する. 具体的には, 最適パス {OP2, OP1, OP5} 上の適用例は, 正の訓練例とし, そのパスから外れて行くような適用例は負の訓練例とする. しかし, この評価が常に正しく行なわれる保証はない. 今の場合は, Fig.2.2 の OP2 の適用例は最適パス上にあるので正の訓練例になる.

一般化部は批評部から渡された訓練例をもとにヒューリスティックを生成する. 洗練途中のヒューリスティックの条件部は, バージョン空間 [Mitchell 78] で表現される. バージョン空間とは正の訓練例をすべて含み, 負の訓練例を1つも含まない概念記述のすべての集合である. 原理的には, このヒューリスティックのバージョン空間は, 外延 (extension) 的に表現可能であるが, 要素の数が無数にあるので現実的には無理である. そこで, 現在の正の訓練例だけをすべて含む最も特殊な要素の集合 : S (これは多くの場合正の訓練例そのもの) と, 最も一般的な要素の集合 : G だけを記憶しておき, バージョン空間はその2つの集合に挟まれた部分として表わす. Fig.2.2 の場合, 最も一般的なヒューリスティックはもとの OP2

の条件部なので，それがGとして設定される．次に，Sには正の訓練例：∫ 3xcos（x）dxが設定される．つまり，SとGは記述言語で表現可能な訓練例と矛盾しないヒューリスティックのすべての集合の範囲の区切りである．よって，さらに正の訓練例が与えられれば，Sはそれらを含むように一般化されるし，負の訓練例が与えられればGはそれらを含まないように特殊化されていく．そして，SとGが徐々に接近していき，ヒューリスティックがより明確になっていく．

Fig.2.3は前述の訓練例を解いた後のOP2に対するバージョン空間を表わしている．このOP2のバージョン空間は，次の新たな練習問題（例えば，∫ 3xsin（x）dx）を解くことによって更新される．Fig.2.4はこの新たな練習問題によりバージョン空間がどのように更新されるかを示している．更新の際に行われる一般化／特殊化は，Fig.2.5に示す概念記述言語のハイアラーキに誘導されて成される．ここでは，Sのcos（x）は新たに

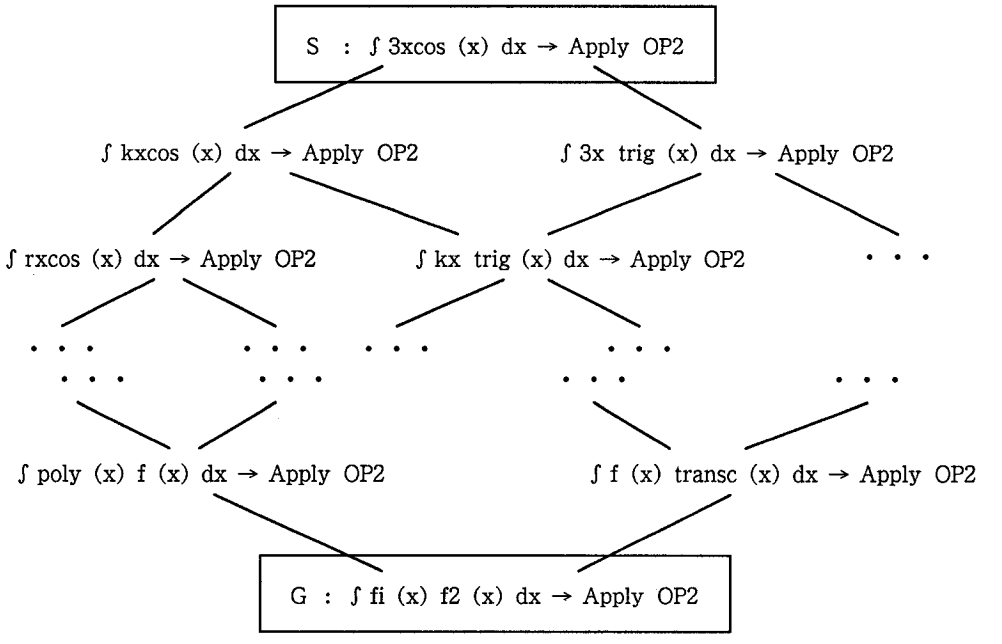


Fig.2.3 バージョン空間

sin（x）が正の訓練例として得られたことにより，Fig.2.5においてそれらを枝としてもつ上位概念であるtrig（x）（三角関数）に一般化されている．また，Gでは負の訓練例を含まないようにg1とg2に特殊化される．このFig.2.5の概念ハイアラーキが帰納学習におけるバイアス（先験的偏向：一般化のための意図的な仕掛）となっている．

2. 1. 3 LEXの限界（帰納的学習の限界）

LEXが正しく学習を行い，有用なヒューリスティックを得るには，一般

ヒューリスティックのバージョン空間 ：

S ： ∫ 3xcos (x) dx→Apply OP2 with u = 3x, dv = cos (x) dx

G ： ∫ f1 (x) f2 (x) dx→Apply OP2 with u = f1 (x),

新しい訓練例：

正の訓練例
∫ 3xsin (x) dx→Apply OP2 with u = 3x, dv = sin (x) dx

負の訓練例
∫ 3Xsin (x) dx→Apply OP2 with u = sin (x), dv = 3x dx

更新されたバージョン空間 ：

S ： ∫ 3x trig (x) dx→Apply OP2 with u = 3x, dv = trig dx

G ：
g1 : ∫ poly (x) f2 (x) dx→Apply OP2 with u = poly (x), dv = f2 (x)
g2 : ∫ f1 (x) transc (x) dx→Apply OP2 with u = f1 (x), dv = transc (x)

Fig.2.4 ヒューリスティック空間の更新

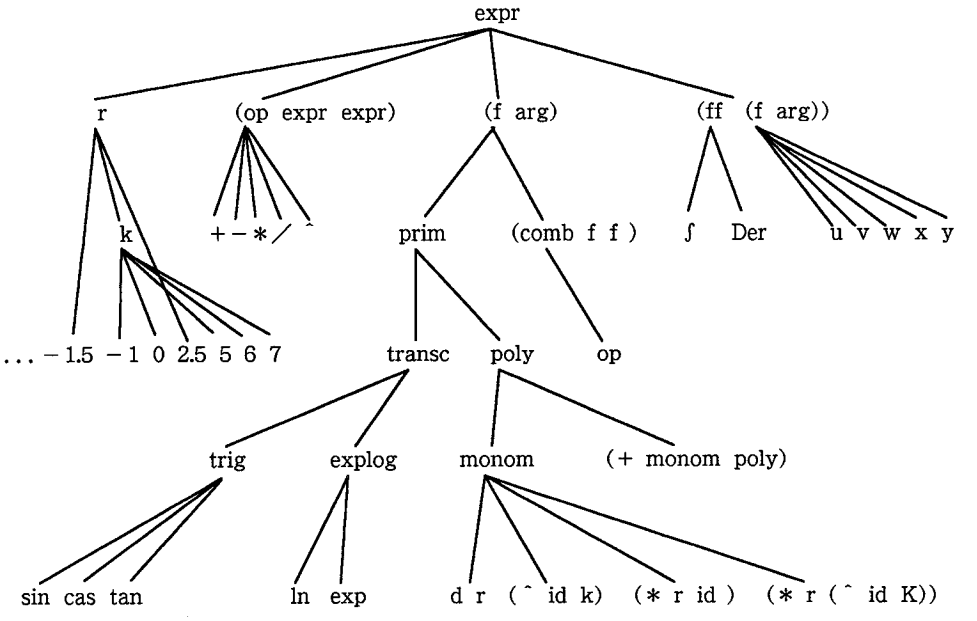


Fig.2.5. LEXでの概念記述言語の文法

的で正確な静的概念ハイアラーキが与えられることが前提となっている. しかし, 事前にこのような概念ハイアラーキを設計する事は非常に難しく, 極言すると十分に一般的で正確な概念ハイアラーキが存在するのかという疑問もある. 実際, LEXシステムで与えられたFig.2.5の概念ハイアラーキで一般化を行うと, 間違った一般化が行われる場合があることが報告されている [Mitchell 81]. その一つの原因は, Fig.2.5の概念ハイアラーキは, いわゆる数学分野の一般的な分類で, 不定積分におけるオペレータによって各関数がどのように変形されるのかということを考慮したものではないからである. そのような対象領域に依存しない概念ハイアラーキではまったく根拠の無い一般化：帰納的飛躍（inductive leap）が必ず起こる. また, いま対象としている問題領域において必要な概念をすべて含む概念ハイアラーキを設計すると, オペレータ個々のヒューリスティック

生成に本当に必要な概念にたどり着くまでに，そのオペレータにとっては関係のない概念をたどっていくことが必要になり，学習の効率が低下することも考えられる．

以上の概念ハイアラーキに関する問題を克服するために，概念ハイアラーキの動的更新を目指したLEXのサブシステム：STABB［Utgoff 86］において2つの手法が用いられた．ヒューリスティックによって概念ハイアラーキのノードを自動生成する方法と解決過程で適用されたオペレータの条件部を後方連鎖させることによって新しい概念記述を生成する方法である．前者の手法はかなり難解なヒューリスティックによって，概念ハイアラーキを修正することが可能であるが，若干ad-hocな印象を受ける．それに対して後者は後方連鎖により一つの訓練例から根拠のある概念記述を生成可能で，この手法を用いたバージョンはLEX2［Mitchell 86］と呼ばれている．この後方連鎖により一般化手法が，後に「説明に基づく一般化」と呼ばれるようになる．

2. 2 説明に基づく学習（Explanation-Based Learning）

本節ではここ数年演繹的学習の代名詞と化している「説明に基づく学習：EBL」について説明していく．EBLに関する研究は，初期の個々の分野におけるもの［Minton 84］［DeJong 82］［Mahadevan 85］［Mooney 86］から，最近のより統一的なもの［Mitchell 86］［DeJong 86］へと変遷してきた．ここではまず，その背景について述べ，さらに具体例としてMitchellらの「説明に基づく一般化：EBG（Explanation-Based Generalization）」について詳述していく．

2. 2. 1 背景

学習システムの最も本質的な能力は，一般化である．前述のLEXのように類似に基づく学習：SBLを行うシステムでは，類似に基づく一般化：SBG (Similarity-Based Generalization) が成される．このSBGは前節でも述べたような概念ハイアラーキに誘導されて一般化が行われが，正確なハイアラーキを設計することは困難であり，そこで根拠の無い一般化が生じてしまう．例えば，正の訓練例として {1, 2, 3} が与えられたとしよう．この訓練例をすべて含む概念は，素数，自然数，整数，正の整数，小数，正の小数，正の有理数，有理数，実数，正の実数，4以下の素数，3以下の自然数，1以上3以下の整数など数多く存在する．通常は，これらの概念をすべてハイアラーキのノードとすることはなく，今対象としている問題領域で必要と思われるものだけを選択する．また，概念間のリンクの張り方も事前に設定される．このように概念ハイアラーキはどのような概念をシステムが学習すべきかをシステム設計者が事前に想定した結果であるので，バイアス（先験的偏見）と呼ばれることもある [Utgoff 86]．しかし，概念ハイアラーキは偏見を含んでいるので，常に正しい一般化を導くとは限らないし，すべての場合を考慮した正確なハイアラーキを事前に設計することは非常に困難である．例えば，{1, 2, 3} は本当は正の整数に一般化されるべきなのに，概念ハイアラーキに正の整数の概念が見落とされて設定されておらず，整数という概念だけがある場合は，この訓練例は任意の整数に一般化されてしまう．このような無根拠な一般化を克服するために，提案されたのが「説明に基づく一般化：EBG」である．

EBGではなぜ根拠のある一般化が可能なのかという理由を簡潔に言う

と，EBGはその名の通り訓練例がなぜ目標概念の正の訓練例であるのかを説明し，その説明が成り立つ範囲で一般化を行うからである．この説明がEBGの一般化の正当性を保証する．言い換えると，EBGは説明により，個々の訓練例について正しい一般化の誘導を与えるわけである．その結果，EBGでは一つの訓練例からでも説明をつけることにより一般化可能である．以降，MitchellらのEBGについて説明していく．

2. 2. 2 MitchellらのEBG［Mitchell 86］

MitchellらはEBGの統一的枠組みを定義した．EBGは以下の4要素を入力として，目標概念の操作可能な記述を出力する．具体例として，Table 2.1にカップの例［Mitchell 86］を示し，それを参考に説明していく．この例では，視覚システムをもつロボットがカップの概念を学習するという状況を設定している．

Given :

・目標概念 : CUP (x) ↔LIFTABLE(x) ∧STABLE(X) ∧OPEN－VESSEL(x)

・訓練例 : OWNER (OBJ1,EDGER), PART－OF (OBJ1,CONCAVITY1),
IS (OBJ1,LIGHT), ・・・

・領域知識 :
IS (x,LIGHT) ∧PART－OF (x,y) ∧ISA (y,HANDLE) →LIFTABLE (x)
PART－OF (x,y) ∧ISA (y,BOTTOM) ∧IS (y,FLAT) →STABLE (x)
PART－OF (x,y) ∧ISA (y,CONCAVITY)
∧IS (y,UPWARD－POINTING) →OPEN－VESSEL (x)
・・・

・操作性基準 :
概念記述は，訓練例を記述するのに用いられる構造的特性を表わす述語(LIGHT,HANDLE,FLATなど）で記述されなければならない．

Table 2.1 CUPの一般化の例

・目標概念（Goal Concept）：学習されるべき概念の記述.

Table 2.1 ではカップ（CUP）が目標概念で，その記述は持ち上げられて（LIFTABLE），安定していて（STABLE），上方が開いている（OPEN-VESSEL）ものである．重要な点はこの目標概念を記述する述語と訓練例を記述する述語は性質が異なるということである．目標概念の方は主に機能的（functional）な性質を表わす述語で記述されるが，訓練例は構造的（structural）な性質を表わす述語で記述されるというように，述語の抽象度レベルが違う．これはカップを認識する際に，視覚システムが直接的に認識できる情報が構造的であり，その情報から推論された結果が機能的であることに対応している．

・訓練例（Training Example）：目標概念に含まれる正の訓練例.

EBG は正の訓練例だけから一般化を行う．Table 2.1 ではカップである OBJ1 の構造的な性質を表わす述語で記述されている．

・領域知識（Domain Theory）：訓練例がなぜ目標概念の正の訓練例であるのかを説明するための知識.

具体的にはルールやファクトで記述されている．前述のように，訓練例を記述する述語と目標概念を記述する述語は異なるので，領域知識を用いて定理証明により結びつけられる．この証明が「説明（explanation）」であり，そのトレースを「説明木（explanation tree）」という．Table2.1 ではLIFTABLE, STABLE, OPEN-VESSEL の定義などがが領域知識となっている．例えば，ある物体が軽くて（LIGHT），その一部（PART

－OF）に取っ手（HANDLE）があれば，それは持ち上げられる（LIFTABLE）というように，LIFTABLEを定義している．このように，領域知識は目標概念の記述述語と訓練例のそれとを関連付ける役割がある．

・操作可能性の基準（Operationality Criterion）

EBGの出力である概念記述は操作可能な述語で記述されなければならないという規定である．操作可能とはパフォーマンスシステムにとって容易に認識可能であるという意味で，カップの例では視覚システムが容易に認識できる述語，つまり訓練例を記述している述語が操作可能な述語になる．よって，目標概念は訓練例を記述している述語で記述されなけらばならない．この操作可能性の基準は，説明過程においては説明木をどこまで展開するかという基準になる．

この操作可能性の基準は重要である．もしこれがないと極端な場合，入力された目標概念の記述がそのままEBGの出力になってしまうことも起こり得る．この意味からEBGは操作可能でない概念記述を操作可能化（operationalization）する，つまり一種の知識の変換を行うものだともいえる［Keller 87b］．

次に，どのようにして上記4つの入力から操作可能な概念記述が得られるかを説明していく．EBGは「説明」と「一般化」の2ステップからなる．

1）説明：訓練例が目標概念の記述をどのように満たすかを領域知識を用いて証明することにより，説明木を構成する．操作可能性の基準により，この説明木のリーフは操作可能な述語でなければならない．

Table2.1の例に対して構成された説明木をFig.2.6（a）に示す．また，Fig.2.6（b）は訓練例の記述を表わしているが，そのうち囲まれている領域がカップという目標概念を記述するために必要な述語である．この述語は説明木のリーフに対応する．このように訓練例の記述の内，概念記述に関係の無い述語を根拠をもって排除できるところが，EBGの一つの特長となっている．

2）一般化：説明木のルートから各領域知識の条件部を満たす範囲でノー

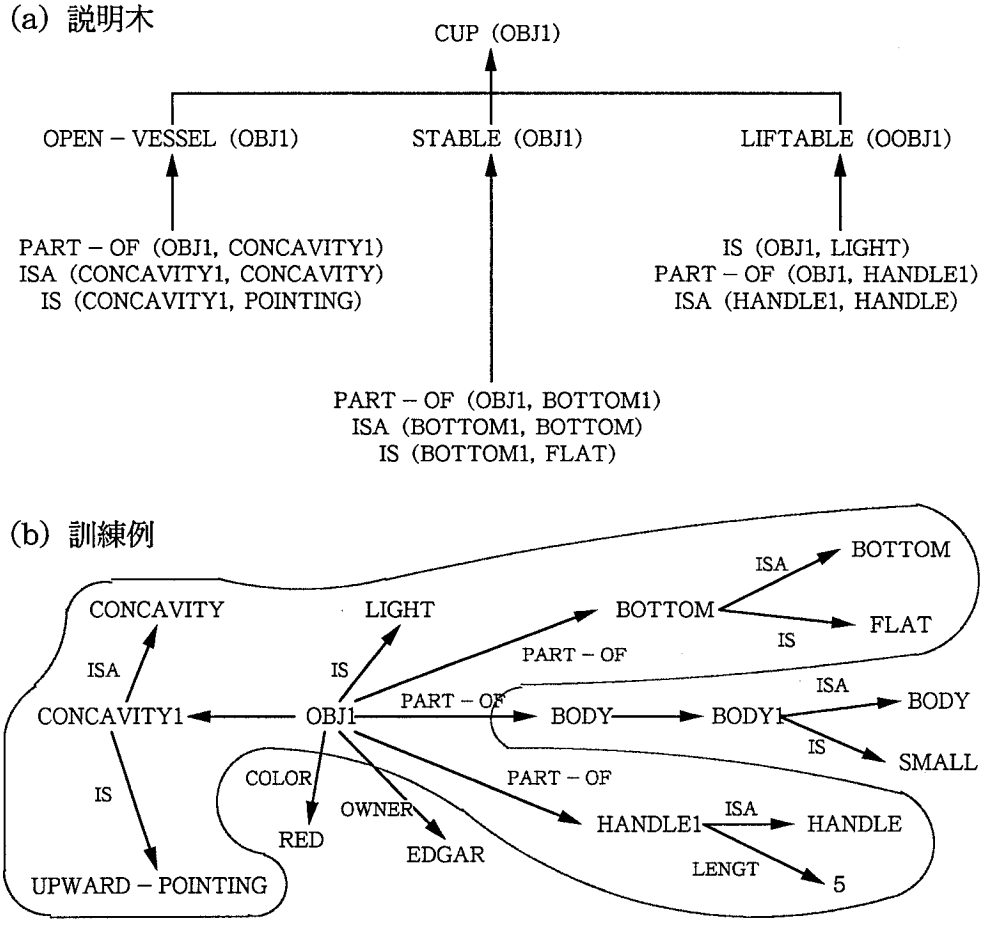


Fig.2.6 CUPの説明木と訓練例

ドの引数を変数に換えて行くことにより一般化を行う．この例では最初に説明木のルートであるCUP(X) がルール：LIFTABLE(OBJ1)∧STABLE (OBJ1)∧OPEN-VESSEL(OBJ1) →CUP(OBJ1) 上で後方連鎖される．その結果，LIFTABLE (x) ∧STABLE (x) ∧OPEN-VESSEL (x) はCUP (x) を推論するための十分条件であると決められる．後も同様に，後方連鎖が行われ，以下のような操作可能性の基準を満たす目標概念記述が出力として得られる．

(PART-OF(x, xc) ∧ISA(xc, CONCAVITY)∧IS (xc, UPWARD-POINTING)
∧PART-OF(x, xb) ∧ISA(xb, BOTTOM) ∧IS(xb, FLAT)
∧PART-OF(x, xh) ∧ISA(xh, HANDLE) ∧IS(x, LIGHT)) →CUP(x)

説明に基づく学習のもう一つの大きな流れとして，説明に基づいて特殊化も行うDeJongらのEBL [DeJong 86] があるが，PiLプロジェクトとの関連があまり無いのでここでは割愛する．

EBGにおける最大の問題点は，EBGは完全かつ十分な領域知識の存在を前提にしていることである．この前提が成り立たないとき，EBGは正確な一般化あるいは一般化自体を行うことが不可能になる．この領域知識が不完全な場合は以下のように分類される [Mitchell 86]．

・不十分な領域知識の問題（The Incomplete Theory Problem）

前述のカップの概念のように非常に単純な例において十分な知識を与えることは比較的容易であるが，株式市場や天候の予測などの複雑な問題では十分な知識を与えることはまずできない．それは現象間の因果関係が明確でないことに由来する．

・処理しにくい知識の問題（The Intractable Theory Problem）

十分な知識が与えられた場合でも，その知識の抽象レベルが適切でないと計算機上では扱いにくい場合がある．例えば，ある物体が他のものを安定してのせることができるという概念：SAFE-TO-STACKを説明するのに量子力学に関する領域知識は明らかに十分な知識であるが，SAFE-TO-STACKの説明には抽象度のレベルが適していないので，量子力学による説明は厳密すぎて無意味な部分が多くなる．このように目標概念に対応した抽象レベルの知識が与えられる必要がある．

・一貫性のない知識の問題（The Inconsistent Theory Problem）

これは領域知識が競合する場合の問題である．もし，同一の概念を説明する複数の知識があれば，複数の説明木が得られ，その結果複数の目標概念記述が得られてしまう．このとき，どの目標概念を選択するかが問題になる．このような状況は，デフォールト値の知識と厳密に計算を行う知識の双方が領域知識として存在する場合を考えれば，容易に想像できる．この問題についてはさきの処理しにくい知識をもからめて，対処法がいくつか提案されている［Braverman 88］［Ellman 88］［Bennett 87］．

また，EBLの新しい方向として，SBLとの融合が考えられる．この研究には，EBLで得られた複数の概念記述をSBLで統合する方法［Kedar-Cabelli 87］や，逆にSBLで得られた結果をEBLで検証する方法［Lebowitz 85］［Numao 86］などがある．

2. 2. 3 Kellerのoperationalityと META-LEX

KellerはMitchellらによる従来の操作可能性の定義を以下のようにより厳密に再定義した［Keller 87a］．ここでは学習システムと問題解決を行

うパフォーマンスシステムを仮定している.

[従来の操作可能性の定義]：概念記述がある概念に含まれるインスタンスを認識するために効率的に使用できればその記述は操作可能である.

[Kellerの操作可能性の定義]：概念記述が以下の2つの条件を満たすとき，その記述は操作可能である.

1）使用可能性（usability）：その概念記述がパフォーマンスシステムにとって使用できる.

2）有効性（utility）：その概念記述がパフォーマンスシステムによって使われたとき，明記された目的を満足するようにパフォーマンスが改善される.

この二つの定義は重なり合う部分も多いが，従来の定義は「パフォーマンスシステムにとって使用可能な概念記述は，必ずパフォーマンス改善に有効な概念記述である」という暗黙の前提があるのに対し，Kellerの定義は使用可能性と有効性を独立したものとして考えそれぞれを評価するところと，有効性を「目的」に依存するものとして捉えているところが決定的に異なる.

上記の操作可能性の定義に基づいて，LEXが対象とした不定積分の分野でオペレータ適用のヒューリスティックを学習するシステムがMETA-LEX［Keller 87b］である．META-LEXの入出力を以下に示す.

＜入力＞

1）操作可能でない目標概念の記述（Non-operational Target Concept Description）：ここではLEXと同様に以下に示すUSEFULという，あるオペレータの適用が有効である問題状態を表わす概念記述が与えられる.

```
USEFUL(move) ⇔ (WITH ((succ (EXECUTE move)))
        (OR  (SOLVED succ)
             (SOLVABLE succ
                       (DIFF & maxdepth  (MOVEDEPTH move)))))
```

この記述は，「あるオペレータを適用した結果（succ）が解決された状態であるか，またはある制限された範囲（ステップ数）内で解決可能（SOLVABLE）な状態であれば，その適応は有効（USEFUL）である」という意味である．また，moveは＜operator, state, binding＞というリストで表わされ，bindingに示されているようにoperatorをstateに適用することで実行（EXECUTE）される．この記述は操作可能ではなくこれを操作可能な記述に変換する，つまり，操作可能化（operationalization）がMETA-LEXのタスクである．

2) 領域知識（Domain Theory）：目標概念を記述している操作可能でない述語（EXECUTE, SOLVED, SOLVABLEなど）の定義．例えば，”積分記号が取れればSOLVEDである”，”ワンステップで解けるか，書き換えた次の状態がSOLVABLEであるならSOLVABLEである”．

3) パフォーマンスシステムの記述（Performance System Description）：「META-LEXのproblem solverであるSOLVERは縦型前向き探索でUSEFUUL概念に含まれる適用を優先する」という記述．

4) パフォーマンスの目的（Performance Objectives）：SOLVERが達成すべき目標．具体的には，下記の2つを満足すること．

・効率レベル”t”（Efficiency Level ”t”）：与えられたすべての訓練例をcpuタイム：t以下で解ける．

・有効比率”p”（Effectiveness Persent ”p”）：与えれた訓練例のp

%以上を解ける.

5) パフォーマンスの環境記述 (Performance Enviroment Description) : SOLVERに与えられた訓練例の集合.

＜出力＞

・操作可能な目標概念記述 (Operational Target Concept Description) : 与えられたパフォーマンスの環境記述においてSOLVERのパフォーマンスの目的を満たすことができるような, つまり操作可能な概念記述.

META-LEXはUSEFULの記述を一般化／特殊化していき, そのあと実際にSOLVERで訓練例を解き直し, パフォーマンスの目的を満たしているか検証するという, 生成-検査法 (generate-and-tset) で操作可能なUSEFUL概念記述を探す. 一般化／特殊化はEBGを用いるのではなく, TRUIFY, FALSIFYという2つの変形だけで行われる. この変形はTRUIFYは概念記述を構成している述語をT (常に真) に, FALSIFYはF (常に偽) に置き換えることである. Fig.2.7にそれぞれの変形の例を示す. この例からわかるようにTRUIFYは一般化を, FAILSIFYは特殊化を行うことに対応する. また, USEFULを構成している述語のすべ

```
USEFUL0 (MOVE) ⇔ (WITH ((SUCC (EXECUTE MOVE)))
                        (OR  (SOLVED SUCC)
                             (SOLVABLE SUCC
                                       (DIFF $ MAXDEPTH
                                             (MOVEDEPT MOVE)))))
→ FALSIFY → USEFUL1 (MOVE) ⇔ (WITH  ((SUCC  (EXECUTE MOVE)))
                                     (OR (SOLVED SUCC)
                                         F ))
→ TRUIFY → USEFUL2 (MOVE) ⇔ (WITH  ((SUCC  (EXECUTE MOVE)))
                                    T ))
→ FALSIFY → USEFUL3 (MOVE) ⇔ F
```

Fig.2.7 TRUIFY,FALSIFYによる一般化

てがこのFALSIFY，TRUIFYの対象になる．このような一般化／特殊化をした後，SOLVERで訓練例を解き直すことによりパフォーマンスの目的が達成されたいるかどうかを調べる．もし，達成されていれば操作可能な目標概念記述が得られたことになり，達成されていなければさらに一般化／特殊化，そして実際の問題解決による検証を続ける．SOLVERの問題解決過程で，個々のオペレータの適用例について，cpuタイム，適用回数，成功した適用回数などが記録され，これが一般化／特殊化の指標となる．ここで重要なことは，USEFULがTRUIFY／FALSIFYされることにより一般化／特殊化されるとT／Fに置き換えられた述語を調べる必要がなくなるので，その分効率が上がるが，反面必要以上に一般化／特殊化される危険があることである．特に過度に特殊化された場合はいままで解けていた問題が解けなくなることが起こる．そのためにパフォーマンスの目的として，効率と有効比率の両方が与えられているのであり，実際に問題解決して目的を満たしているか否かを調べることにより，過度の特殊化／一般化を防いでいる．このように，META-LEXにおける一般化はEBGではない．

いま，SOLVERからFig.2.8のようなデータが得られたとする．このデー

```
SUBEXPRESSION # 118    TYPE : EXISTENTIAL  SUPEREXPR PTR : 7
  TIME :  40.6 CPU SEC    EVALUATED : 240    TRUE : 0    FALSE : 240
  SUBEXPR :  [EXISTS BINDING IN  (BINDING 'OP23 STATE)
                    (WITH  ((SUCC  (LEX - APPLY 'OP23 BINDING)))
                       (OR  (SOLVED SUCC)
                            (SOLVABLE SUCC  (SUB1 DEPTH))))]
QUANTIFIED BODY PTR :  119
```

Fig.2.8 OP23に関するデータ

タはOP23の適用が240回行われ，すべて失敗したということを表わしている．ここで適用が失敗したとは，その適用によって展開された探索木が結局解決状態にいたらなかったことを意味する．つまり，OP23の適用は全くの無駄ということなので，Fig.2.9のSOLVABLE－1のようにSOLVABLE概念を修正する．これはSOLVABLEを構成している各オペレータの適用を表わす部分（(OR [EXISTS ・・・）からOP23の適用の部分をFAILSIFYすることにより特殊化し，OP23の適用がSOLVABLEかどうかを検証する必要がないようにしている．そして，さらにUSEFULを以下のようにSOLVABLE－1を用いたものに修正する．

USEFUL (MOVE) =

(WITH ((SUCC (EXECUTE MOVE)))

```
SOLVABLE－1 (STATE) =
    (AND (GT DEPTH 0)
        (OR [EXISTS BINDING IN (BINDINGS 'OP1 STATE)
                (WITH ((SUCC (LEX－APPLY 'OP1 BINDING)))
                    (OR (SOLVED SUCC)
                        (SOLVABLE SUCC (SUB1 DEPTH))))]
                    •
                      OP2－OP21
                    •
            [EXISTS BINDING IN (BINDINGS 'OP22 STATE)
                • • •]
             F－1     % F－1は常にFAIL
            [EXISTS BINDING IN (BINDINGS 'OP24 STATE)
                • • •]
                    •
                      OP25－OP30
                    •
                    •))
```

Fig.2.9 FALSIFYにより特殊化されたSOLVABLE概念

(OR (SOLVED SUCC)

(SOLVABLE-1 SUCC (DEFF & MAXDEPT

(MAXDEPTH MOVE)))))

以上のことにより，ヒューリスティック：USEFULの適用においてOP23の適用は避けられ，その分パフォーマンス効率が上がる．さらに今与えられている訓練例の集合については，OP23の適用がすべて失敗に終わっているので，SOLVABLE概念をSOLVABLE-1のように修正したとことにより，解ける訓練例の個数の比，すなわち有効比率が低下することはない．よって，パフォーマンスの目的を二つとも満たす方向に，つまり操作可能化の方向に目標概念が再表現されることになる．どの述語を一般化あるいは特殊化するかの判断はヒューリスティックによって行なわれる．

以上のような操作可能化により，META-LEXは異なった訓練例の集合（パフォーマンスの環境記述），異なったパフォーマンスの目的（具体的には，tとpの値を変える）に対してそれぞれを満足するような操作可能なUSEFUL概念記述を生成できることが実験的に検証されている[Keller 87b]．また，META-LEXは操作可能化に関する研究ではあるが，EBGに関する研究ではないことをつけ加えておく．

META-LEXの最大の問題点は，一般化／特殊化がTRUIFY，FAILSIFYを用いた非常に単純なものであるために，操作可能なUSEFULあるいはSOLVABLE概念の正確な記述を得るのは難しく，あくまでも近似的な目標概念記述しか得られないことである．前述のようにMETA-LEXの一般化／特殊化はEBGを用いたものではなく，どちらかというと帰納的一般化の最も単純なものであるといえる．この一般化／特殊化の手法が改善

されない限り，常に正確な記述を得ることはできないと考えられる．

もう一つのMETA-LEXの問題点は，操作可能性の検証に非常に計算コストがかかることである．META-LEXの操作可能化は基本的には生成-検査法なので効率は悪いし，さらに実際にパフォーマンスシステムに解かせることにより検証は非常にコストがかかる．もちろん，ヒューリスティックによる一般化／特殊化の枝がりや検証において無駄な解き直しは行わないなどの処置はされているが，それでも多くの計算量が必要と考えられる．とくに与えられた訓練例の数が非常に多い場合は，検証の計算コストは多大なものとなるであろう．

2. 3 Soar

Soar [Laird 84] は心理学におけるチャンキング仮説 [Miller 56] に基づく汎用学習機構である．チャンキング仮説とは，人間はチャンクという表現形式で環境から知識を獲得し，短期記憶に格納できるチャンクの数は 7 ± 2 に制限されており，それが人間の認知能力の限界を説明すると言うものである．以下，Soarについて述べていく．

2. 3. 1 Soarの問題解決

Soarの問題解決は目標指向（Goal-Oriented）の探索を実行するプロダクションシステムによって行なわれる．この探索の行なわれる問題空間は，問題の状態とそれを変化されるオペレータの集合であり，状態及びオペレータを総称してオブジェクトと呼ぶ．目標指向の問題解決とは，現在の状態から次に達成されるサブゴールを常に設定し，そのサブゴールに至

るオペレータを探していき，最終状態までたどり着くというものである．現在の状態からサブゴールに適用できるオペレータの候補が複数個ある場合は，それらの中から一つを選択しなければならない．この選択のための知識を探索制御知識と呼ぶ．Soarではこの探索制御知識が完全であれば，適用すべきオペレータを一意に決定できるので問題解決は迅速に行なわれる．しかし，一般的には初期段階でこの知識は不完全であるため，オペレータを一意に決定できずに行き詰まってしまう．そこで，この行き詰まりの状態において，オペレータの候補を評価するというサブゴールが新たに設定される．このような探索がオペレータについてだけではなく，オブジェクト全体について行なわれる．

具体的な例をFig.2.10で説明する．まず，Aにおいてサブゴールを達成

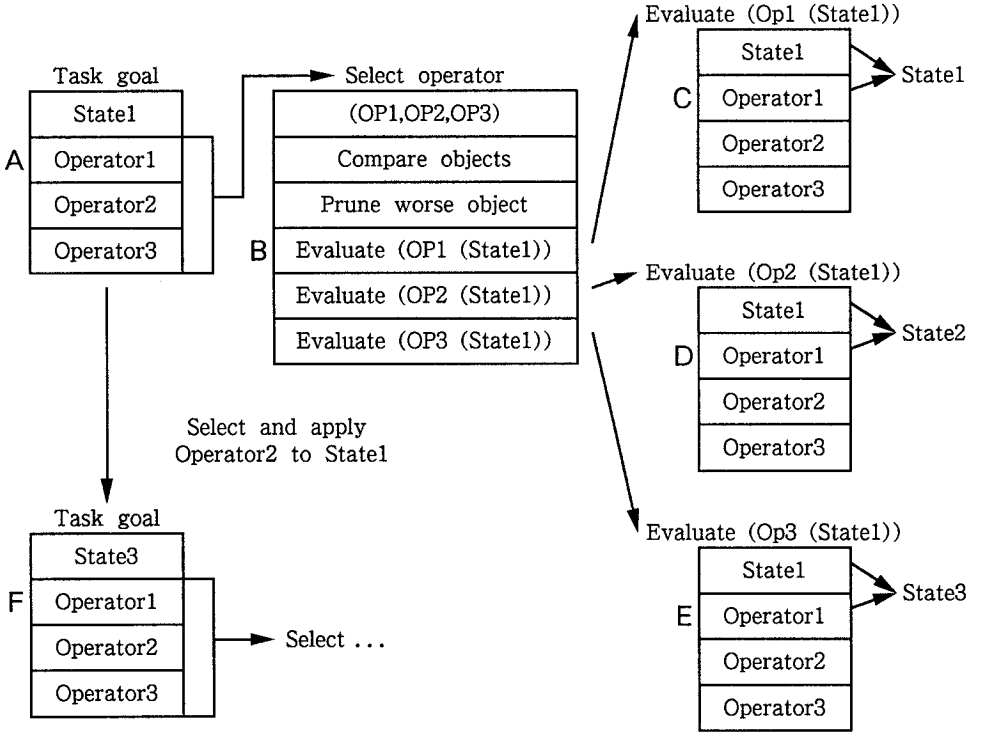


Fig.2.10 Soarでのサブゴール構造

するオペレータの候補がOperator1～3の3つあるが，探索制御知識が完全でないため，一つを選択できないとすると．次にこの行き詰まった状態を切り抜けるため，各オペレータ候補の評価を行なうサブゴール：Bが設定される．さらに，Bにおいて各オペレータの評価（Evaluate（Op1(State)）など）が探索制御知識を用いて行なえないときは，さらに各オペレータを実際にState1に適用して調べるというサブゴール：C，D，Eが設定される．これらの展開されたサブゴールを達成した結果，オペレータ2が最適なオペレータとして選択され，AのState1にオペレータ2が適用され，Fに移る．

この行き詰まったときのサブゴールの達成の方法をできるだけ一般的な形式で記憶しておき，後の類似の問題の解決に役立てようというのが，次に述べるチャンキングによる学習である．

2. 3. 2 Soarにおける学習

Soarにおける学習は，人間が新しいルールを直接追加する方法と，問題解決において行き詰まりをどのように解消したかをSoar自身が自動的に学習する方法がある．ここでは後者の手法について述べていく．

まず，Soarの学習は問題解決過程で行き詰まりが起こったときに開始される．前述のように行き詰まりの状態ではそれを解消するためにサブゴールが設定されるが，Soarはこのサブゴールが解決される間にどのような情報が利用されたかを収集・要約して記憶し，後に同様の行き詰まりがおきた場合に過去の解消法を用いて直接的に解決し，問題をより迅速に解く．例えば，複数のオブジェクトの候補から一つを選択するサブゴールの解決からは，探索制御知識を学習する．

学習された知識はプロダクションルールで表現される．このプロダクションルールの条件部と結論部をどう構成するかであるが，Soarはサブゴール毎にそれが生成された原因とそれを達成するために必要な情報を蓄えておき，それらから条件部を生成する．このとき条件部中で特定オブジェクトに対する検査ではなく不特定オブジェクトに対する検査の項目がある場合は，そのデータが変数化され一般化される．また，サブゴールを達成したした結果で結論部が構成される．この変数化以外に，サブゴールを解決するために参照された情報以外は条件部から取り除くことで一般化が行なわれる．このような一般化手法を陰的一般化（Implicit Generalization）と呼ぶ．

2. 3. 3 Soarの適用例

ここではSoarを8パズルに適用した例［Laerd 84］を紹介する．8パズルの問題状態は3×3のグリッドにおいて，1〜8までの数字の書いてある8個のタイルの配置で表わされる．オペレータはブランクを上：U，下：D，左：L，右：Rに移動させるものである．探索制御知識は，現在の状態から最終状態までに動かすタイルの数をもとに，どのオペレータを選択的に適用すればよいかを決めるものである．

しかし，探索制御知識が不十分な場合は，候補選択用のサブゴールが設定され，学習が行なわれる．Fig.2.11の（a）は学習をしないSoarがある問題：task1を解決した過程である．ここでは，オペレータ評価のために枝分かれが頻繁に行なわれているのがわかる．そして，（b）は学習を行ないながらの問題解決を表わす．この図では，前半のオペレータ評価が学習されて後半に用いられることにより，後半において枝分かれが減少して

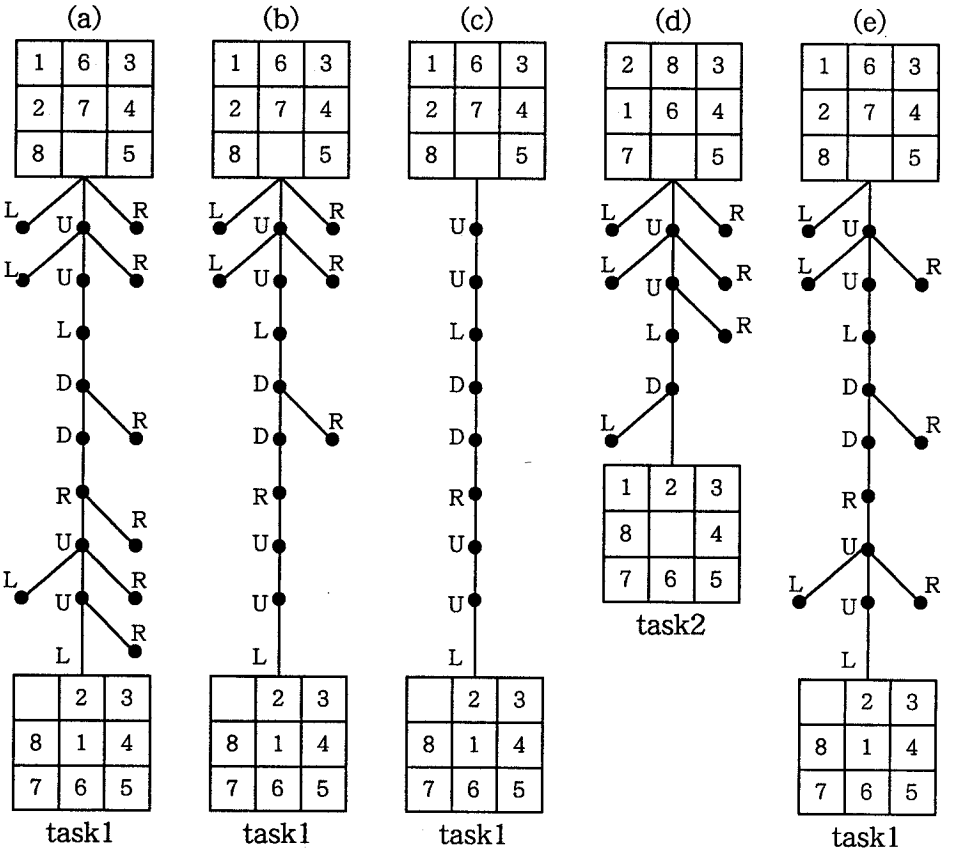


Fig.2.11 Soarでの8パズルの学習

いる様子がわかる．さらに，（c）は（b）を行なった後に再び同じ問題：task1を解かせた場合であり，枝分かれがまったく無くなっている．（d）は別の問題：task2を学習させた過程を示し，（e）は（d）の後にtask1を解かせた過程を表わしている．別の問題：task2を解くことによって得られた知識が一般化されることにより，task1の問題解決に役立てられることがよくわかる．

2. 3. 4 Soarの問題点

Soarの最大の問題点はその一般化手法である．陰的一般化手法ではどの要素を一般化（変数化）するのかを的確に決定できる保証が無いので，

当然過度の一般化（over-generalization）が起こり得る．その場合，誤った知識が学習されることになるが，それに対する処置がないことが問題である．

2. 4 マクロオペレータ学習

PiLプロジェクトにおける重要な研究対象として，マクロオペレータの学習がある．本節ではマクロオペレータに関する研究を先駆的研究であるMACROPSから最近のものまでをながめていく．

2. 4. 1 STRIPSのMACROPS

STRIPS［Fikes 71, 72］は古典的かつ最も有名な問題解決システムであり，現在でも広く使われている．FikesらのSTRIPSに関する論文は2つある．最初の論文［Fikes 71］は，問題状態（ロボットの行動する世界）を述語で表わし，各オペレータの適用可能性を導出法によって求めるという論理的側面，また現在の問題状態と目標状態との差異を縮小するオペレータを探すという一般問題解決器：GPS［Ernst 69］を具現化したシステムとしての側面などを強調したものであった．そして，ここで言及する2つ目の論文［Fikes 72］はSTRIPSを強調している．ここではSTRIPSは過去に解いた問題の解決過程を三角表という表現形式で残しておき，次にまた同じような問題を解決する際に過去の解決過程をマクロオペレータとして用いて効率的な問題解決を行うというものである．このSTRIPSにおけるマクロオペレータをMACROPSと呼ぶ．三角表の生成方法及びその表からMACROPSを抽出方法については，種々

の解説があるので，ここでは割愛する．また，STRIPSにおける一般化手法は，現在盛んに研究の行われている「説明に基づく一般化」に多大な影響を与えた．

STRIPSにおけるマクロオペレータ：MACROPSの最大の問題点は，過去の解決過程を重複する部分を除いて，すべて残していくことである．これにより，比較的少数の問題を学習させた場合でも，膨大なマクロオペレータが生成されてしまう．その結果，その多数のマクロの探索に多くのコストがかかってしまい，学習無しの問題解決システムよりも効率が低下してしまうことが起こってしまう [Minton 85]．この問題については，第6章で詳しく述べる．

2. 4. 2 R. KorfのOperator Decomposability

R. Korfは8パズル，ルービックキューブなどでマクロオペレータを用いて探索無しに効率的に問題解決を行なう手法を示した [Korf 83, 85, 87]．Korfはマクロオペレータを「すでに達成されたサブゴールを覆すことなく，さらに一つのサブゴールを達成するオペレータシーケンスである」と定義し，この特性を持ったマクロを自動的に生成する手法を提案した．例として，8パズルにおけるマクロオペレータをTable2.2に，8パズルの最終状態をFig.2.12に示す．ここでは，ブランクは0である．問題状態は現在何番のタイルがFig.2.12の何番の位置にあるかで表わす．Table.2.2は，すでに最終状態にあるタイルを動かすことなく，あるタイルを最終状態に移動させるオペレータシーケンス，つまりマクロオペレータを表の形で表わしている．U，D，L，Rはそれぞれ上下左右にタイルを移動させるオペ

タイルの初期位置	タイルの番号 0	1	2	3	4	5	6
0							
1	UL						
2	U	RDLU					
3	UR	DLURRDLU	RLUR				
4	R	LDRURDLU	LDRU	RDLLU RDRUL			
5	DR	ULDRUR DLDRUL	LURD LDRU	LDRULU RDDLUR	LURD		
6	D	URDLDRUL	ULDDRU	URDDLU LDRRUL	ULDR	RDLLUUR DLDRRUL	
7	DL	RULDDRUL	DRUUL DRDLU	RULDRDL ULDRRUL	URDLULDR	ULDRUR DLLURD	URDL
8	L	DRUL	RULL DDRU	RDLUL DRRUL	RULLDR	ULDRRU LDLURD	RULD

Table 2.2 8パズルにおけるマクロオペレータ

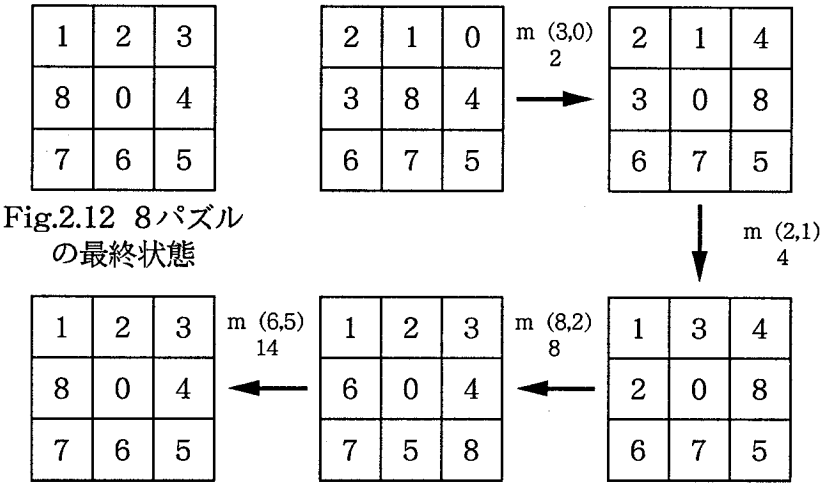


Fig.2.12 8パズルの最終状態

Fig.2.13 マクロによる問題解決

レータである. 表の行はいま最終状態に移動しようとしているタイルがFig. 2.12のどの位置にあるのかを表わし, 列は移動させるタイルの番号である. マクロ問題解決システム (Macro Problem Solver) は, Table.2.2の列を0 (ブランク) から順に最終状態に移動させていく. この際, 既に最終状態にあるタイルはとばす. 前述のKorfの定義を満たすマクロオペレータがあれば, まったく探索無しに効率よく問題を解決することが可能である. 8パズルでは181440個の問題がわずか35個のマクロオペレータを用いて探索無しに解くことができた [Korf 83].

具体例として, Fig.2.13にTable2.2のマクロを使った解決過程を示す. 図中, m (i, j) とその下の数字はそれぞれTable2.2のi行j列のマクロオペレータとそのマクロがいくつのオペレータで構成されているかを表わす. まず, ブランク : 0が最終状態に移動され, 次に1, 2, 5が順に移動された結果, すべてのタイルが最終状態になる. この例では, 合計28ステップかかる解決過程がマクロにより4ステップに軽減していることがわかる.

しかし, どんな問題領域においてもKorfの定義を満たすマクロオペレータが存在するとは限らない. そこでKorfはそのようなマクロが存在するための条件を数学的に定義した. それがOperator Decomposability [Korf 85] である. また, 実際にオペレータを適用していきマクロオペレータを自動生成する方法も述べられている. この自動生成において, 単純な縦型探索, 双方向探索 (bi-directional search), マクロオペレータの合成などの手法がある.

Korfのマクロオペレータの問題点は, Operator Decomposability

が成り立たない問題領域ではマクロが得られないことである．これは彼のマクロオペレータの定義がかなり狭義なものなので，それを満たすマクロが得られれば探索がなくなり，非常に強力であるが，マクロの生成できる問題範囲はかなり限定されるというTrade-offがある．また，マクロの自動生成にかなりの計算コストがかかると思われる．さらに重要なことは，マクロオペレータが解法例から学習されるのではないことである．Korfの手法ではオペレータと最終状態が得られれば，マクロオペレータを生成可能である．この方法の欠点はマクロオペレータが非常にたくさんあり，通常はそのほんの一部しか必要ない場合でも，すべてのマクロオペレータを生成してしまうことである．これに対して，解法例からの学習では解法例中で実際に使われたことのあるマクロオペレータしか学習されない．

2. 4. 3 Mintonの選択的マクロオペレータ学習

Mintonは多数あるマクロオペレータの候補の中から，選択的にマクロを生成すつことの重要性を指摘した［Minton 85］．そして，選択的マクロ学習を行なうシステムをインプリメントし，無差別にマクロを生成するSTRIPSとの比較を実験的に行い，その有効性を検証した．彼のシステムが学習するマクロオペレータは次の2種類である．

a) S-MACRO：過去の解決履歴を残しておき，共通する部分シーケンスをマクロとする．

b) T-MACRO：サブゴールに相互作用がある場合に有効なマクロ．探索木が枝分かれする点の近傍で数ステップのマクロが生成される．

さらに，これらのマクロには上限が設定されており，それを越える場合は最も古いマクロから強制的に削除されていく．Mintonはこのようにマ

クロを選択的に学習できるシステム，学習無し問題解決システム（学習無しSTRIPS)，選択無しにマクロを学習するシステム（MACROPSを学習するSTRIPS）の3つのシステムで同じ問題を学習させる実験を行なった．その結果，選択無しに学習するシステムは少ない問題でも非常に多くのマクロを学習してしまい，ついには学習無しシステムよりも効率が低下してしまうという興味深い現象が確認された．また，選択的学習システムは最後まで学習無しシステムよりも高い効率を維持できた．

MintonのS-マクロ生成についての問題点として，放題な過去の履歴をすべて残しておくのか，解決履歴の共通部分を容易に抽出できるのか，また少数の解法履歴からはS-マクロは生成できないのではないかということが挙げられる．

第３章　問題解決における戦略知識の獲得

一般に問題解決システムの知識ベースを構築しようとする専門家（いま対象としている問題領域の専門家であって問題解決システム構築の専門家ではない）が容易にかつ明確にシステムに与えることのできる知識は，問題の状態を変化させる公式的かつ基本的な書換えルール（以後，基本オペレータ）である．しかし，通常この基本オペレータはその条件部が一般的過ぎるというように精練されておらず［Mitchell 83a］，これをそのまま用いていたのでは前向き後向き推論に関わらず全く無意味な適用が数多く行なわれ，探索空間が爆発的に拡大してしまい解にたどり着けない場合が多い．例えば，今回対象としている各種方程式の解法においても，適用可能な基本オペレータを全て適用して前向き探索していたのでは，探索木が爆発的に展開されてしまう［Bundy 85］．また，数式処理の基本オペレータ（例えば，結合法則，分配法則など）は条件部のみならず結論部も一般的過ぎるので，STRIPS［Fikes 71, 72］流に目標状態から各基本オペレータの結論部を調べることにより関連するオペレータ（relevant operator）［Fikes 71］を探すという後向きの戦略でも探索空間が爆発してしまい解決不可能である．方程式解法の一般的過ぎる基本オペレータの例として，「等式の両辺から任意の整式を引く」というオペレータがある．この基本オペレータはあらゆる等式に適用可能であり，かつ結論部の任意の整式は無限個考えられるのでその適用結果は無限にあり，このオペレータの適用ですでに探索木の爆発的展開が起こってしまう．さらにこのオペレータは方程式を解くために不可欠な「移項」に必ず必要なため，その条件部を精練しなければ問題解決は不可能である．

また，基本オペレータは非常に断片的な知識なので，解に至るまでに必要なステップ数が非常に多い問題を解く場合でも，ワンステップ毎に次に適用すべきオペレータの探索を行わねばならない．このような問題解決は，複数の基本オペレータのシーケンスを一度に適用して展開ステップ数を減らしている人間の問題解決と比べるとはるかに効率が悪い．よって，問題解決システムを実用に耐えるものにするには，基本オペレータの適用を制御する知識や特定の基本オペレータシーケンスの連続的適用を促すような知識，つまり知識（基本オペレータ）に関する知識であるメタ知識が必ず必要となる．問題解決で探索を制御するこのようなメタ知識を本論文では「戦略知識」と呼ぶことにする．この戦略知識は，具体的には条件部が精練された基本オペレータや複数の基本オペレータを一つに合成したマクロオペレータなどがある．

従来の問題解決システムでは効率的なパフォーマンスを実現するために専門家が戦略知識を事前に与えていたわけである．しかし，決まりきった公式的な基本オペレータとは異なり，戦略知識を問題解決システムの知識ベースにいれるには以下のような問題がある．

問題1：「専門家は自分が知っていると思っている以上のことを知っている」といわれるように，戦略知識はいわゆるヒューリスティックスなので専門家自身がその知識を自覚しにくい，あるいはまったく意識していない場合がある．よって，専門家がどれほど自己のもっている戦略知識を対象化できるのか，またそれらの知識のすべてを列挙できるのかが疑問である．

問題2：専門家が自分の戦略知識を対象化できたとしても，知識ベースの専門家ではない問題解決の専門家がはたしてそのヒューリスティックスを問題解決システム内の知識表現形式に正確に変換し，表現できるのかが

問題である.

一般には以上のような問題に対して知識工学者 (knowledge engineer) が問題解決システムと専門家の間のインタフェースとして介在し, 専門知識の知識獲得を人手によって行なっているが, この作業は多大な労力を要する. 現在問題解決システムの実用的一例であるエキスパートシステム構築のために, 種々の支援ツールが出回っているにも関わらず, この知識獲得を包含したものはほとんど見られず, それがボトルネックとなっている. このような背景のもとに知識獲得の半自動化・自動化の要望がますます高まっている. 以上のようにこの戦略知識獲得における問題に対処することは非常に重要な意味を持つ. これまでETS [Boose 84], MORE [Kahn 85] などの知識獲得自動化の研究が行なわれているがそれらはいずれも, 専門家側ですでに対象化され問題解決システム内の知識表現にちかい形式に変換されている知識のみをどのようにうまく採取するかという手法を提案してきたに過ぎない. そこでは問題2の一部が解決されているに過ぎず, 問題1については手つかずのままである.

ところで基本オペレータ以外に専門家が容易に入力可能なデータとして実際に専門家が対象領域の問題を解いた履歴があげられる. これはいま問題解決システムを構築しようとする専門家は, 対象領域における問題解決の専門家なので当然のことである. 本章で提案する手法は, まず正確な基本オペレータが与えられ, 次に専門家が対象領域の問題を解決した履歴である解法例が与えられ, 学習システムがその解法例を分析することにより, 基本オペレータの条件部の精練とマクロオペレータの生成を自動的に行なうことにより戦略知識獲得の自動化を実現するというものである (Fig.3.1). この手法により専門家自身が自分のもつヒューリスティック

スを対象化して，問題解決システムの知識表現に変換する必要はなくなり，前述の2つの問題が解決される．このような専門家の問題解決過程を観察して，できるだけ専門家に干渉せずにヒューリスティックの獲得を行なうシステムは「学習見習いシステム（Learning Apprentice System）」と呼ばれ，LEAP [Mitchell 85]，VILLA [Watanabe 86] などの稼働システムが報告されているが残念ながらまだ実用段階までは至っていない．

本章では，以上のようなパラダイムのもとに構築された問題解決における戦略知識の獲得を行う学習システム：PiL（Paradigm－based inference Learner）について詳細に述べていく．まず，PiLシステムの概要を，次に各モジュールの働きについて説明し，最後に各種方程式の解法学習における実験を行い，PiLシステムの学習能力を検証する．なお，PiLシステムの特徴として以下のものがあげられる．

1）説明に基づく一般化による基本オペレータの精練．

2）オペレータの重み付けによる絶対オペレータの抽出．

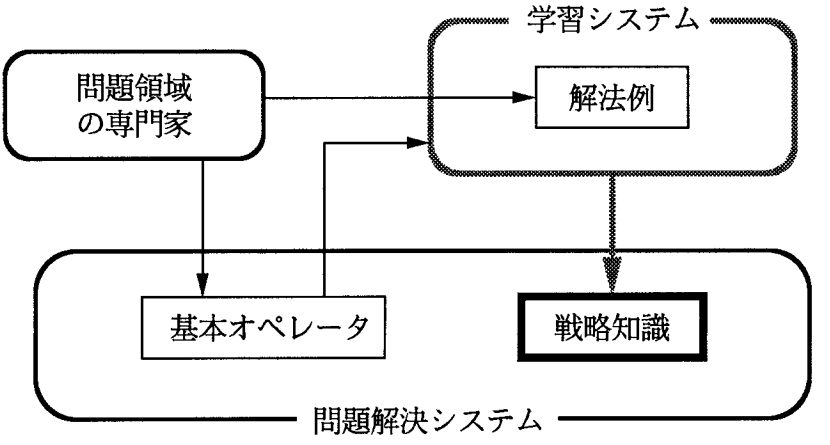


Fig.3.1 戦略知識の獲得

3) 完全因果性によるマクロオペレータの選択的抽出.

このうち, 2), 3) は従来の機械学習研究にはない新しい知見である.

3. 1 PiL システム概要

PiL は人間の教師に基本オペレータと模範的な解法例を与えてもらい, その解法例を肯定例として一般化することにより, 書換えルールを精練し戦略知識の学習を行う. 学習後の PiL は学習前よりも問題解決の効率が飛躍的に向上する. なお, PiL は問題解決において基本的に横型探索を行い, オペレータの条件部の精練と後述する知識の階層構造だけで推論の制御を行っている. Fig.3.2 に PiL の構成図を示す. システムは大まかに, 問題解決モジュールと学習モジュールの 2 つのモジュールからなる.

PiL の入力は, 基本オペレータ, 終了条件ルールと解法例である. 終了条件ルールとは, 問題が解決されたことを判別するルールで目標状態をその条件部にもつ. まず, 問題解決の専門家である教師が基本オペレータと終了条件ルールを最初に与える. 次に, 教師より問題 (今の場合は方程式) が与えられ, 最初は PiL の問題解決モジュールが基本オペレータを用いて問題状態を変化させていき独力でその問題を解こうとする. しかし, 前述のように学習前の状態では戦略知識が全くないので, すぐに探索木の展開が爆発してしまい解決には至らない. そこで PiL は一定の探索空間を調べ尽くしても解にいたらない場合, 教師に解法例の入力を要請し, 教師の助力によりインタラクティブに問題を解決する. この際, 教師により与えられた解法例を PiL のもっているオペレータのシーケンスに詳細化する. こうして解法例が得られると学習モジュールがインスタンスであるその解

法例をEBGで一般化していく．その結果得られる一般化された問題状態を，対応する基本オペレータの条件部に連言として付加したものが，基本オペレータ個々に関する戦略知識であるヒューリスティックオペレータである．さらに，得られたヒューリスティックオペレータに重み付けを行うことにより単独だが高い優先度をもつ絶対オペレータを抽出し，次に完全因果性という基準を用いて特定のオペレータシーケンスを取り出し，マクロオペレータを生成する．そして，最後に冗長な知識を取り除いて知識の整理をするというサイクルを繰り返す．以降，各々のモジュールの動作

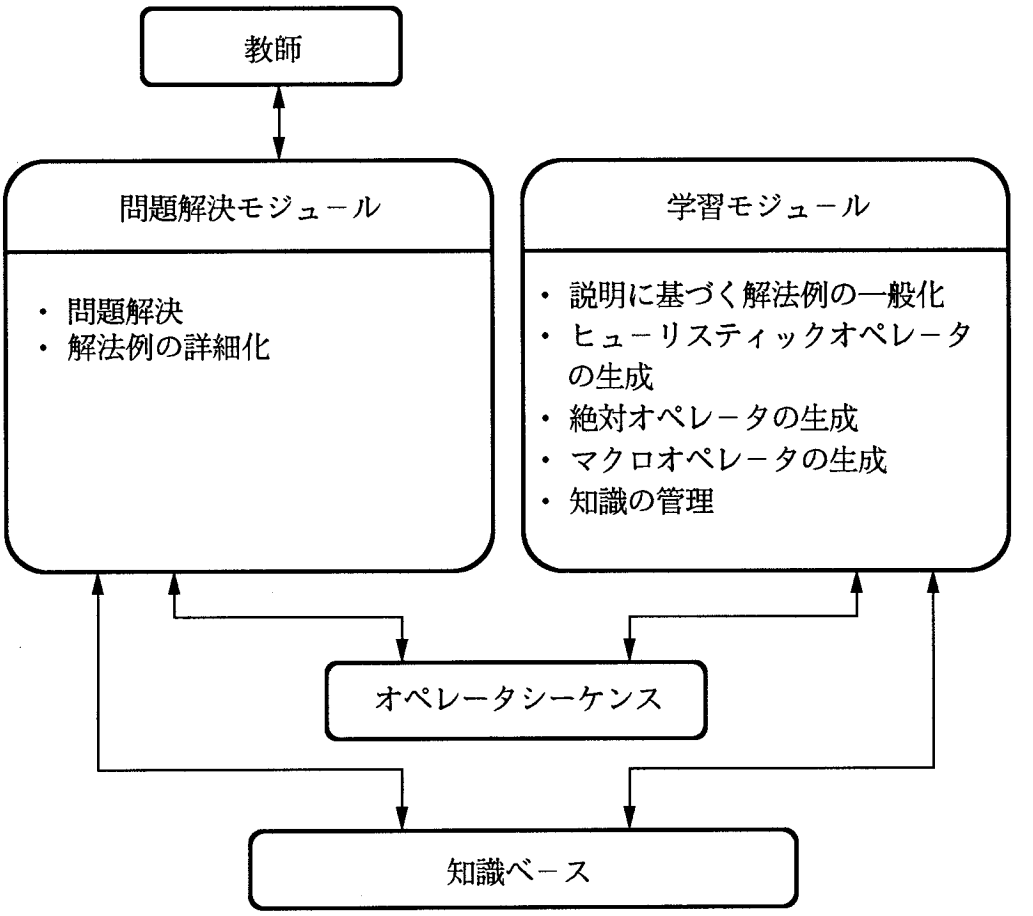


Fig.3.2 PiLのシステム構成

原理を詳細に述べる.

3. 2 PiLの問題状態表現と知識表現

まず最初にPiLシステムはprologでインプリメントされているので基本的な表現（例えば，大文字は変数を表わすなど）はprologに準拠していることを明言しておく.

3. 2. 1 問題状態表現

PiLは設計段階で方程式の解法の分野を対象とすることが考慮されていたので，その問題状態表現は方程式の問題状態，つまり数式に依存している．PiLは教師により挿入辞記法（infix）で与えられる数式を内部表現である接頭辞記法（prefix）に変換しており，さらにこの接頭辞記法をリスト構造で表現している．しかし，オペレータの適用の処理を容易に行えるように，リスト構造に若干の修正をした．一般的な接頭辞記法のリスト表現はFig.3.3のAに示すような［演算子，引き数1，引き数2］という3項のリストである．しかし，この表現形式はオペレータのマッチングの際に頻繁に行なう「同じレベルにある項を探す」などの「項」に関する処理にはあまり適していない．ここでいうレベルとはネストの深さではなく，

$3 * x - 5 * x + 3 = 10$

A : [= ,[+, [−, [∗, 3, x], [∗, 5, x]], 3], 10]

B : [=, [+, [∗, 3, x], [∗, − 5, x], 3], 10]

Fig.3.3 問題状態の内部表現

数式の項の深さのことで例えばFig.3.3だと3＊x，－5＊x，3 が同レベルである．つまり，項がリスト表現上で明示的に表わされておらず，また項のレベルとネストの深さが対応していない．よって，表現形式自体からは直接的に項が認識しずらくその分マッチングを行うインタプリタに負担がかかる．そこで少しでもインタプリタの負担を減らし処理速度を速めるために，項を明示的に表わすように改良したFig.3.3のBのようなリスト表現を用いている．この表現は［演算子，任意個の引き数］という形式で，この引き数が項を表わしている．また，ネストの深さと項のレベルが一致しており，同レベルの項を探すのに都合がよい．さらに，このリスト表現では項を明示的に示すためにマイナス演算子は使用せず，項の係数にマイナスを掛けるか，係数のない場合は－1を掛けて表わす．この表現によって基本オペレータで項の係数がプラスの場合とマイナスの場合の区別がなくなり，プラスの場合のオペレータだけを与えておけば良いことになる．例えば，A＊B＋A＊C→ (B＋C)＊Aを与えれば，A＊B－A＊C→ (B－C)＊A というオペレータは与えなくてよい．

3. 2. 2 オペレータの表現形式

Fig.3.4に今回PiLに実際に与えた方程式の分野での基本オペレータの一部を，またAppendix1に今回のPiLの方程式の解法学習実験に用いた全ての基本オペレータを示す．この例のように基本オペレータは下のような構造をもつ書き換えルールで，マクロオペレータを含めPiLの扱う全てのオペレータはこの形式で統一して表現されている．

ルール名 ： 条件部→結論部，条件の拘束リスト，結論の拘束リスト

オペレータの各部の意味であるが，条件部が現在の問題状態（ここでは

数式）とマッチングがとれると，その例示化（instanciate）された条件部の構成要素（例えば，項，係数）についての拘束条件である条件の拘束リストの要素の述語がすべて真であるかを調べ，真であれば結論の拘束リストの述語を満たすように結論部を例示化して，最後に現在の問題状態での条件部とマッチングのとれた部分を例示化された結論部に書換え，新しい問題状態をつくる．

次に各部の構成要素のついてくわしく説明していく．まず，オペレータ中の大文字のアルファベットは変数を表している．条件部，結論部ではリスト構造に変換された数式の構造的なマッチングのみを行い，そのマッチングされた各要素についての拘束条件を各拘束リストが表す．条件部，結

```
rule010  :  LS = RS → LS + A = RS + A,  [],  []
rule030  :  LS = RS → LS／A = RS／A,  [],  [not__zero (A)]
rule210  :  A * B + A * C → A * (B + C),  [],  []
rule220  :  0 + A → A,  [],  []
rule230  :  0 * A → 0,  [],  []
rule270  :  R1 + R2 → 0,
              [real__number (R1), real__number(R2),
               equal(R1,(− 1) * R2)],  []
rule280  :  A／A → 1,  [not__zero (A)],  []
rule300  :  R1 + R2 → R3,
              [real__number (R1), real__number (R2)],
              [equal (R3,R1 + R2)]
rule320  :  R1／R2 → R3,
              [real__number (R1), real__number (R2),
               not__zero (R2)],
              [equal (R3,R1／R2)]
rule340  :  (A * B) ／C →  (A／C) * B,  [],  []
rule1000 :  1 * AL = R → finish,
              [alphabet (AL),real__number (R)],  []
```

Fig.3.4　基本オペレータ

論部の数式はシステム内部では前節で述べたリスト表現で表わされているが，Fig.3.4では分かりやすいように挿入辞記法で書いてある．以後，オペレータの条件部，結論部は挿入辞記法で表わすことにする．

次に各拘束リスト中の述語は，すべて［述語記号，引き数1，・・・］というリストの形式で表現されている．各拘束部内の述語には以下のようなものがある．

1）概念述語：対象がある概念に属しているか否かを返す1引数の述語

- real_number／1：引き数が任意の実数であれば真．
- alphabet／1：引き数が任意のアルファベットであれば真．
- not_zero／1：引き数が0以外の任意の整式（実数だけも含む）であれば真．
- not_one／1：引き数が1以外の任意の整式（実数だけも含む）であれば真．
- plusp／1：引き数が任意の正の実数であれば真．
- minusp／1：引き数が任意の負の実数であれば真．

2）演算述語：引き数間の拘束を満たすように変数を例示化する．

- equal／2：第二（または一）引数の数式を評価した結果を第一（または二）引数に単一化できれば真．
- log／2，＊＊／2，＊／2，plus_minus_sr／1：これらは通常，関数と呼ばれるもので引き数に対数・指数・積・平方根の演算を行なった結果の値を返す．

3. 2. 3 概念のハイアラーキ

例えばminuspはreal_numberに包含されるというように，概念述語

間にはFig.3.5のような概念の包含関係がある. 拘束部の概念述語は冗長ではあるがその述語の表す概念を包含しているより一般的な概念述語は全て記述する. つまり, Fig.3.5でルート以外の先祖ノードをすべて記述する. 例えば, minup(X) だけではなくreal_number(X)& minusp (X) と書くようにする. このような記述は暗示的明示的にかかわらず概念間の階層関係を必要とする. このことは「説明に基づく一般化にはいわゆる概念ハイアラーキが必要ない」というEBGの利点を損なうものと考えられるかも知れない. しかし, この記述は後に述べる一般化の最適経路選択や知識の管理におけるオペレータ間の処理のみにとって必要なものなので一般化には関係していないことを強調しておく.

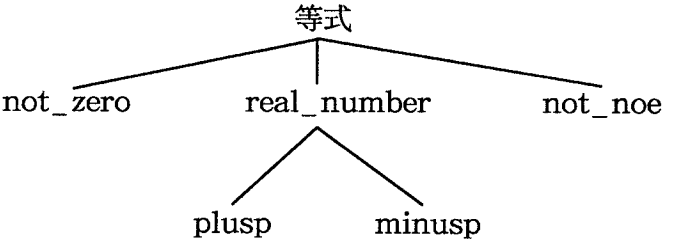


Fig.3.5 概念述語のハイアラーキ

3. 3 知識ベース

各種オペレータで構成されているPiLの知識ベースは, Fig.3.6のように4層からなる階層構造になっている. 以下に各オペレータについて述べる.

1) 終了条件ルール

終了条件ルールは教師により入力として与えられるもので, このルール

が適用されれば問題は解決されたことになる．Fig.3.4ではrule1000がこれにあたる．PiLで用いている全ての終了条件ルールはAppendix1に示されている．これらのルールのうち現在与えている問題の解決状態として望ましいものを一つ最上位の階層に入力として入れている．また，複数個のルールを最上位階層に入れておき，それらのうちの一つでも適用されると問題が解決されたことにするとも可能である．この終了条件ルールは解法例のEBGにおける一般化伝搬の根本になるものなのでその条件部は必要十分に一般的に記述されなければならない．

2）マクロオペレータ

複数の基本オペレータをひとつに合成したオペレータ．このオペレータの適用により基本オペレータでは複数ステップかかる展開をワンステップで行え，探索空間を縮小して効率を大幅に向上することができる．マクロオペレータはPiLの学習モジュールにより自動生成される．

3）絶対オペレータ

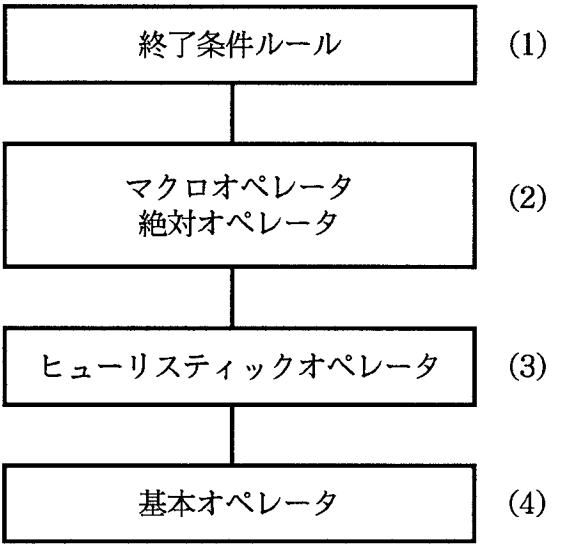


Fig.3.6 オペレータの階層

マクロオペレータとは違い単独の基本オペレータではあるが，マクロオペレータと同等の優先度をもつオペレータで問題解決の効率化に役立つ．典型的なものとして実数演算のオペレータなどがある．この絶対オペレータも後述する手法により，学習モジュールによって自動的に抽出される．

4) ヒューリスティックオペレータ

解法例をEBGにより一般化した結果を基本オペレータの条件部に連言として付加し，無意味な適用を行なわないように基本オペレータを精練したオペレータ．これは個々の基本オペレータについての戦略知識である．

5) 基本オペレータ

最初に入力として与えられる基本的な書換えルール．PiLでは与えられる問題を解決するために十分で正しい基本オペレータが入力として与えられると仮定している．

以上のすべてのオペレータは3.2.2で述べた形式で統一的に表現される．また，まったく学習していない初期状態では知識ベースは基本オペレータと終了条件オペレータだけであり，学習とともに戦略知識であるヒューリスティックオペレータ，絶対オペレータ，マクロオペレータが獲得され蓄えられていく．

3. 4 PiL システムの問題解決

問題解決モジュールでは，prologによって書かれた問題解決システムによる問題解決及び解法例の入力が行われる．

3. 4. 1 問題解決

問題解決モジュールは前節のFig.3.6の階層を（1）から（4）の順序で適用可能なオペレータを探索していく．各階層では横型探索がされ，競合解消をまったく行なわない．競合解消は行なわず，オペレータの階層を最初に設定して固定しておき，あとはどの階層にどのようなヒューリスティックオペレータやマクロオペレータを追加するかでオペレータ適用の制御を行なうということがPiLの設計方針である．ルールインタプリタは上位から順に各階層で適用可能なオペレータを探索していき，適用されるオペレータがまったくない場合に下位の階層の探索を始める．もし，すべての階層で適用可能なオペレータがまったくないときは探索は失敗する．各階層の探索中に適用可能なオペレータが見つかると，それらを現在の問題状態に適用して問題状態の更新を行ない，再び（1）に戻り最初から探索が行われる．つまり，問題状態が変化する毎にそれが終了状態であるかないかのチェックを行なうわけである．そして，一定サイズの探索空間を調べ尽くしても解にいたらない場合や探索空間が爆発した場合のように，PiLが独力で解決できないときは，教師の援助を要求する．

3. 4. 2 解法例の入力

PiLが与えられた問題を自力で解けた場合はその解決結果が「解法例」となり問題解決モジュールに渡される．つまり，解法例は人間の教師が解いたものである必要はなく，問題解決モジュールが解いたものでもよい．なお，PiLは最適な解法例が得られると仮定している．PiLが独力で解けない場合は教師が正しい解法例を教示して行くことによりこの前提は成立する．また，PiLが独力で解いた場合は，問題解決モジュールは横型探

索なのでその解決過程は最短ステップであるという意味で最適な解法例と言える.

しかし, 一般に学習前のPiLは問題を独力で解くことはできない. PiLは一定の探索空間を調べ尽くしても解けない場合や探索空間が爆発的に拡大したとき, 独力では解決不可能であると考え教師に援助を要請する. Fig.3.7で解法例の入力を説明する. この木のルートノード:node0は問題の初期状態である. このnode0からまずPiLは独力でオペレータを適用し一定の探索空間:S1を探索した結果, S1内で解決できなかったとする. この際, PiLは教師に現在の確定している問題状態ノード(今の場合はnode0)を表示し, そのノードの次に来るべき問題状態の入力を教師に求める. そして, 教師はnode1のような問題状態を与えたとする. 新しく問題状態が教師から与えられると, 問題解決モジュールはそれをサブゴールとして現在確定しているノードからそのサブゴールにいたるオペレー

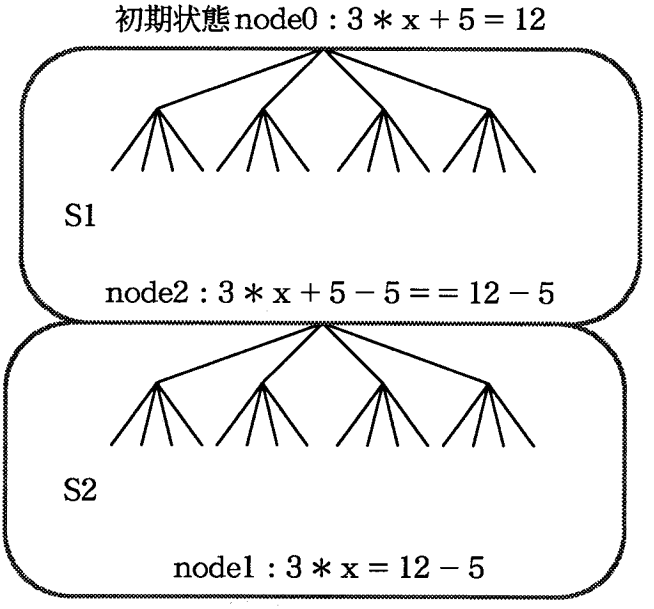


Fig.3.7 解法例の入力

タシーケンスを探す．しかし，通常教師が自然に与えたサブゴールは数ステップの基本オペレータが省略されておりPiLが自力でそのサブゴールにたどりつくことは難しい．今の場合も問題状態：node1はS1内にないのでそのサブゴールたどり着けない．このとき問題解決モジュールは教師にnode0とnode1の問題状態を表示し，この間にあるサブゴールの入力を教師に要請する．すると教師は例えばnode2のような問題状態を入力してくれる．このnode2はS1内に入っているのでnode0からnode2をオペレータで結ぶことができ，サブゴールnode2は達成される．すると残るのはサブゴールnode1である．node2は確定されたノードとなるので，node2からさらに一定の探索空間：S2が調べられる．すると今度はnode1がS2内にあるのでサブゴールnode1も達成され，確定したノードになりそこからさらに探索が行なわれる．以上の探索の繰り返しにより，PiLが独力で解決できない場合でも初期問題状態から解決状態にいたるオペレータシーケンスを教師の助力により得ることができる．

当然のことだが解法例はPiLのもっているオペレータで構成されたオペレータシーケンスでなければならない．上述の方法では教師の助力をかりながらも結局PiL自身が自分の持っているオペレータを適用して問題を解いていくので，この条件は満たされている．また，問題解決モジュールがオペレータの適用により探索木を展開していく際，横型で行なうので同じステップ数で同じ結果のでる展開が生じるが，それらは全て一般化伝搬経路の候補として保持しておき，後の一般化の際に適切なオペレータが選択される．

以上のようにして得られた解法例がFig.3.8である．図中のL1～L16の各行は問題状態を表わし，その各行の間のルールは適用された基本オペレー

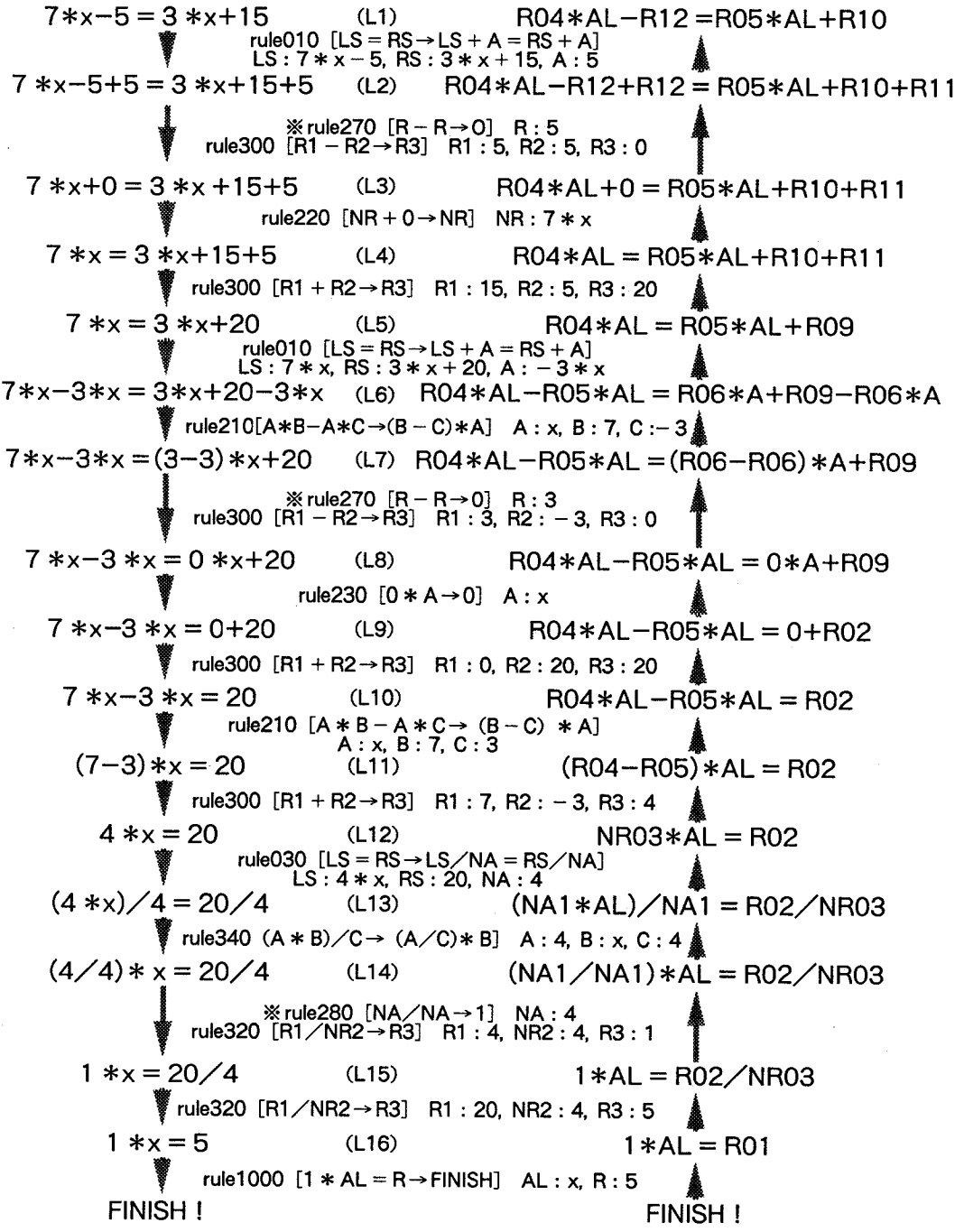


Fig.3.8 解法例

Fig.3.9 一般化の伝播

タを表わす．この基本オペレータは例示化（instantiation）されていないものでり，その右または下に各変数がどのように例示化されたかを示している．またここでの基本オペレータは簡略化のため条件部と結論部だけで記述されており，次のような記号を用いている．

LS：左辺，RS：右辺，A，B，C：任意の整式，R，Rn：任意の実数，
NR：0以外の任意の実数，NA：0以外の任意の整式，AL：任意の変数

ここで重要なことは，解法例中の例示化された基本オペレータはそれぞれ対応する基本オペレータの肯定例であるということである．例えばL1のrule010で考えてみるとrule010：LS＝RS→LS＋A＝RS＋Aの肯定例が7＊x－5＝3＊x＋15→7＊x－5＋5＝3＊x＋15＋5である．つまり，常に保証されているとは限らないが，次にもし同様の問題状態に出会った場合，過去の肯定例と同様の適用を行なうとうまく問題が解けるであろうと仮定できる．過去にうまくいった適用と同様の適用が有効であることは理論的には保証されないが，PiLは戦略知識というヒューリスティックを学習することが目的なので問題にならない．なぜなら，ヒューリスティックはあくまで経験的知識でありその有効性が理論的に保証されなくても，実験的に保証されれば十分だからである．

ところが解法例中の例示化された基本オペレータは肯定例の個々のインスタンスであり，これらをこのまま持っていたのでは次に全く同じインスタンスが来たときにしか使えない．このようなインスタンスを多数蓄えていく「暗記的学習」の手法もあるが，そのような手法は機械学習において最も低級なもので膨大な数の訓練例を与えなければならず，効率が極端に悪い．よって，これらの肯定例をオペレータの適用が有効な範囲で一般化し，他の問題状態のインスタンスに対しても適用できるようにしなけれ

ばならない.

3. 5 戦略知識の獲得

学習モジュールでは, 1) 説明に基づく解法例の一般化, 2) 絶対オペレータの抽出, 2) マクロオペレータの選択的生成, 4) 知識の整理などが行われる.

3. 5. 1 説明に基づく解法例の一般化

学習モジュールはまず解法例の各行を一般化する. 当然その結果, 例示化された基本オペレータも一般化され, この一般化された基本オペレータが個々の基本オペレータに関する戦略知識である「ヒューリスティックオペレータ」である. PiLが行う一般化は機械学習の分野で従来よく行われてきた類似に基づく一般化 [Mitchell 81] [Kilber 83] ではなく, 「説明に基づく一般化 : EBG (Explanation-Based Generalization)」の手法である [Mitchell 83a,83b]. 具体的には, 解法例中のオペレータの条件部が最後の終了条件ルールから後方伝播して行くことにより一般化が実現される. 説明に基づく一般化を用いる利点として, 1) 類似に基づく一般化に必要な全体的かつ統一的な概念ハイアラーキが不要, 2) 無根拠な一般化を避けることができる, 3) 一つの解法例から一般化が可能であり非常に学習効率がよいことなどがある. 例えば, 訓練例の整式に同じ数字が含まれていたとき, 従来の一般化ではこれを根拠も無く同一の変数に一般化するが, 説明に基づく一般化では後々同一であることを必要とする条件部を持つオペレータが適用されているものだけを同一の変数として一

般化する.

説明に基づく一般化を用いて解法例を一般化する過程を具体的な例で説明する. Fig.3.9にFig.3.8に示した解法例が一般化された結果を示す. 図中の各行において, R0nは任意の実数（n の等しいものは同一の実数), A, B, Cは任意の整式, NA, NB, NCは0以外の任意の整式, LS, RSは左辺・右辺, NRは0以外の任意の実数, ALは任意のアルファベット（変数）を表す. Fig.3.8とFig.3.9がLEX2におけるEBGの説明木［Mitchell 86］に相当する.

一般化の伝搬は終了条件から始まる. まず, 終了条件rule1000 の条件部の左辺は1と任意のアルファベットの積, 右辺は任意の実数でよいので, Fig.3.8でのL16の左辺の"x" はALに, 右辺の"5" は任意の実数R01に一般化され, Fig.3. 9のL16のようになる. 次にrule320を逆にたどることにより, L15の右辺の"20" と"4" はR02, NR03に一般化される. さらに, L14ではrule280 により左辺の"4" と"4" が等しい任意の0でない整式 NA1に一般化され, L14はrule340によりL13のようにない る. そして, 次はrule030により伝播が行なわれるわけであるが, rule030は結論部で0でない同一の整式（NA）で両辺を割っているので, これによりL13の両辺の分母であるNA1とNR03が単一化されて, 拘束の強いNR03になる. その結果L12の左辺のALの係数がNR03になっている. あとは同様にL1まで一般化が行なわれていく. ここで重要なのは, 十分な一般化を行なうために, 一般化の伝播は完全に後向きで成されるということである. このことによりオペレータは自分よりも先に適用されたオペレータの条件部の影響はまったく受けないことになる. その結果, 常に解法例の最後のオペレータである終了条件ルールが一般化されることはない. 例

としてL6を見ると，L6の上のrule010の結論部で両辺に同じ整式Aを足しているので，L6でrule010の整式Aとマッチする R05＊AL と R06＊Aが単一化されそうであるが，rule010はL6よりも先に適用されているので，単一化はされない．そして，L5において R06＊A は R05＊AL に単一化される．もし前向きの伝播も許すと，伝播される拘束条件が必要以上に増え一般化が十分に行なわれない．

この一般化伝播の具体的な過程を比較的簡単な解法例であるFig.3.10の例を用いて詳しく説明していく．一般化は以下の3つの処理からなる．

1）解法例の最大一般化

2）解法例の各行の単一化

3）オペレータの拘束条件の割当

以下それぞれの処理について述べる．

1）解法例の最大一般化： Fig.3.10の（a）は解法例を示す．まず，最初に学習モジュールはこの解法例の各行を適用されたオペレータと共に最大限に一般化する．この最大一般化は適用された基本オペレータの構造を残した範囲で等式の辺，項を変数に変換することである．この最大一般化の例をFig.3.11に示し，Fig.3.10の（b）に（a）の解法例を最大一般化した結果を示す．（b）において（a）の1行に対して2行が対応しているのは，適用されたオペレータの条件部と結論部それぞれについてその構造を保持するように最大一般化されているからである．また，複数個のオペレータが適用されているところではそのオペレータそれぞれについて最大一般化が行なわれる．

2）解法例の各行の単一化： このように最大限の一般化が終わると，（a）の1行に対応する（b）の破線をはさんだ行のペアを終了条件ルールから

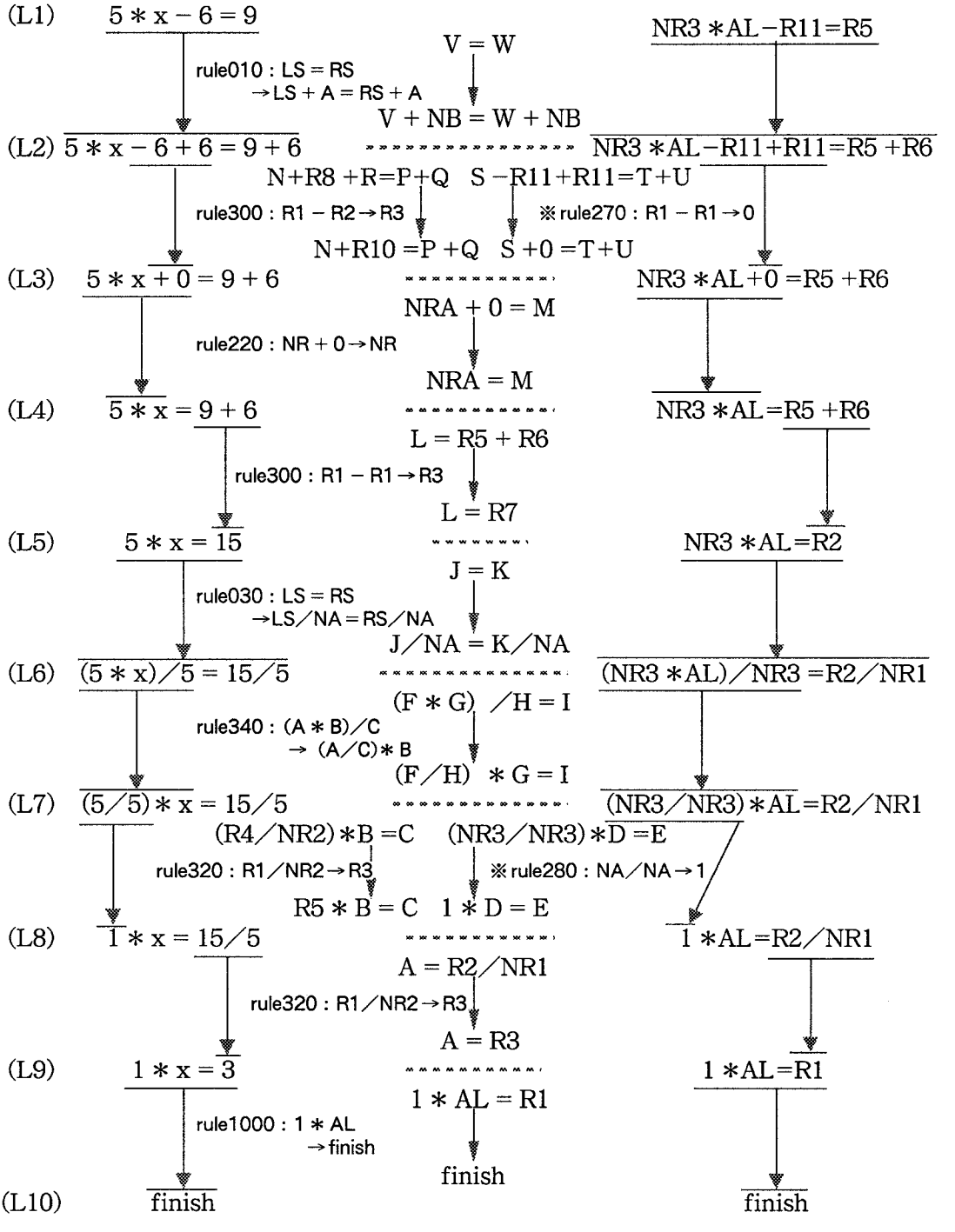


Fig.3.10 一般化の具体的処理過程

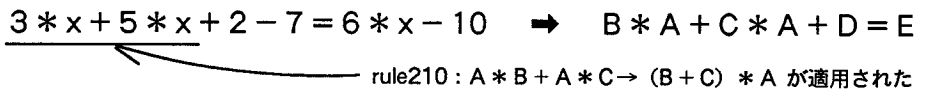


Fig.3.11 最大一般化

単一化して行く．この際前にも触れたように単一化は完全に後向きだけであり，自分より上にある行の単一化の影響を受けることはない．このように一般化の処理において，変数とアトム間，変数と変数間の単一化が頻繁に行なわれることが，システム構築の使用言語として，単一化処理が内部に組み込まれているprologを用いた主な理由である．またここではrule300のような実数どうしの演算を行なうオペレータでは条件部と結論部の変数間の束縛（binding）がないので，実数の四則演算を行なうオペレータにおける一般化伝播のために，「実数は加減乗除について閉じている」という知識を付加的に与えている．さらに，この単一化の過程で複数個の経路の中から適切なものが一つ選択されるが，それについては次節で言及する．このように単一化が行なわれることにより，最大一般化された解法例の各行及び適用されたオペレータが構造的に特殊化される．

3）オペレータの拘束条件の割当：最後に一般化された解法例の各行にその行以降に適用されたオペレータの拘束条件のうち関連するものを各行に割り当てることによりさらに特殊化を行なう．ここで関連する拘束条件とは各行の変数と同じ変数を引数に持つ拘束条件である．

以上で一般化伝播は終了し，その結果がFig.3.10の（c）である．以上の一般化で特徴的なことは，まず解法例を最大限に一般化してから特殊化するということである．この手法は説明に基づく一般化では広く用いられているものである．

こうして一般化された解法例が得られるが，その各行における適用されたオペレータの条件部にあたる部分が，そのオペレータの精練された条件部と考えられるので，適用されたオペレータも同時に一般化されたことになる．こうして得られた精練された基本オペレータを「ヒューリスティックオペレータ」と呼ぶ．このヒューリスティックオペレータは個々の基本オペレータについての戦略知識で，解法例中で適用され例示化された基本オペレータからみると一般化したものであり，もとの基本オペレータからみると条件部を特殊化して精練されたものである．

Fig.3.8の例から得られたすべてのヒューリスティックオペレータをFig. 3.12に示す．➡の左にあるのがもとの基本オペレータで，右にあるのがそれを精練して得られたヒューリスティックオペレータである．この図から基本オペレータがうまく精練されていることがわかると思うが，特にrule010とrule030では「等式の両辺に任意の整式を足す，あるいは割る」という基本オペレータが図のように特殊化され，精練されている．もちろん，これらの基本オペレータが適用されるべきパターンは他にもあるがそれは後にそのパターンがでてきたときに学習される．

rule010 : LS = RS→LS + A = RS + A
　　➡ NR * AL − R1 = R2→NR * AL − R1 + R1 = R2 + R1
rule270 : R1 − R1→0 ➡ R1 − R1→0
rule220 : NR + 0→NR➡ NR * AL + 0→NR * AL
rule300 : R1 + R2→R3 ➡ R1 + R2→R3
rule030 : LS = RS→LS／NA = RS／NA
　　➡ NR * AL = R1→ (NR * AL) ／NR = R1／NR
rule340 : (A * B) ／C→ (A／C) * B
　　➡ (NR * AL) NR→ (NR／NR) * AL
rule280 : NA／NA→1 ➡ NR／NR→1
rule320 : R1／R2→R3 ➡ R1／R2→R3

Fig.3.12 Fig.3.8から得られた
ヒューリスティックオペレータ

3. 5. 2 一般化伝搬経路の選択

3. 4. 2で述べたように解法例の入力の際に複数の基本オペレータの条件部及び結論部の適用範囲（オペレータの適用範囲とは，その条件部または結論部及びそれらの拘束部が内包的に表わす問題状態の集合を言う）の重複により，複数の基本オペレータから同一の結果が得られることがある(Fig.3. 8のL2, L7, L14, Fig.3.10のL3, L8). このとき，学習モジュールは一般化伝播の際にそれらの候補から適切なものを選択する. ここで適切なものとは複数個のオペレータのうち，その例示化されていないもとの結論部の適用範囲がいま単一化されようとしている一つ下の行の適用範囲と最も交わりの大きいものを意味する. 例えば，Fig.3.8のL14ではrule280と320が競合しているが，L15はルールの結論部に"1"を求めているので，結論部がR3という任意の実数であるrule320よりも結論部が1であるrule280の方が適用範囲の交わりが大きい（この場合は一致している）のでrule280が選択される. 他の場合においても同様に※印のルールが選択される. 具体的には各候補のオペレータとそれぞれ独立に単一化を行いその前後で条件部，結論部の変数の個数の変化が少ないもの，また同じ概念述語の個数の多いものがより適切と考えられ選択される.

3. 5. 3 ヒューリスティックオペレータの優先順位
（絶対オペレータの抽出）

次に，得られたヒューリスティックオペレータに優先順位を付ける. ここで問題となるのは，どのような基準で優先順位を決めるかである. 筆者は基本オペレータの条件部の元々の適用範囲とその基本オペレータがEBG

により一般化され精練された後の，つまりヒューリスティックオペレータになった後の適用範囲との交わりの相対的な大きさに着目した．つまり，ある基本オペレータの適用範囲（構造パタン及び拘束条件）をA，その基本オペレータを一般化して得られたヒューリスティックオペレータの適用範囲をBとすると，BとAの比の値：B／Aが大きいヒューリスティックオペレータほど優先されることにする．しかし，この適用範囲間つまり集合の内包的表現の演算であるB／Aを定量的に求めることは難しいので，ここでは特殊な場合のみを扱う．それはA＝Bとなるとき，つまり精練された適用範囲がもとの適用範囲と一致する場合で，この条件が成り立つヒューリスティックオペレータは他のヒューリスティックオペレータよりも優先される．A＝Bという演算はオペレータの条件部が変数・定数の変化なしに単一化でき，かつ拘束部が等しいかを調べることにより，比較的容易に検証できる．このA＝Bの条件を満たすヒューリスティックオペレータを「絶対オペレータ」と呼び，ヒューリスティックオペレータよりも一つ上位のマクロオペレータと同じ階層に移すことで優先順位を上げる．その結果，絶対オペレータは単独のオペレータではあるが，ヒューリスティックオペレータよりも優先的に適用されることになる．ここでのヒューリスティックオペレータの優先順位は絶対オペレータとそうでないヒューリスティックオペレータの二つだけのグループ分けである．このような定義により，人間が方程式解法において優先的に適用している単独の基本オペレータである実数どおしの演算などの基本オペレータなどが絶対オペレータとして抽出される．

また，前節で述べたように複数のオペレータが適用されたときは，その内の最適な一つが選択されるが，その選択された基本オペレータが絶対

オペレータになった場合，当然それ以後は競合した他のオペレータ：OP1よりも絶対オペレータになったオペレータ：OP2の方が優先的に適用されることになる．しかし，将来場合によってはOP2よりもOP1の方が一般化伝播の経路として適当である場合も起こる．よって，そのような場合に対処するために競合して選択されたオペレータが絶対オペレータになったときは，その他の競合したオペレータもすべて絶対オペレータとするという幾分ad－hocな処理を行なう．

3. 5. 4 マクロオペレータの選択的生成

基本オペレータは断片的な知識の集合であるから，それを一般化したヒューリスティックオペレータや絶対オペレータもまた断片的な知識に過ぎない．このようなワンステップ単位のオペレータだけでは，問題解決においてオペレータを展開する際に，細かいステップ毎に次に展開すべきオペレータを探さねばならず非常に効率が悪い．例えば，これらのワンステップオペレータだけを使った方程式の解法はFig.3.8の解法例のようになる．しかしこの図を見ると明らかなように，人間はこのようなワンステップごとの解き方をしておらず，例えば，基本オペレータ（あるいはヒューリスティックオペレータ）だと2ステップかかる変数項どおしの和差などをワンステップで行なっている．つまり，ある特定のオペレータのシーケンスを一つにまとめたマクロオペレータを持っており，それを用いて効率的な問題解決を行っている．そこでPiLも特定のシーケンスをマクロオペレータとすることにより，効率を上げる必要がある．

しかし，マクロオペレータ生成のもととなる特定のオペレータシーケンスがどこから得られるかが問題である．PiLでは解法例という実際に問題

を解いた具体例を想定しているので，その解法例中から特定のオペレータシーケンスを抽出すればよい．だが，ここでもう一つ大きな問題がある．nステップの解法シーケンスについてのマクロオペレータの候補数：MC（n）は，順序関係を崩さない範囲でのすべての組合せの数と考えられるから，MC（n）$=\Sigma_{k=2}^{n}$nCkとなる．例えば，10ステップのオペレータシーケンスから考えられるすべてのマクロの候補数は，MC（10）＝1013個にも昇る．つまり，解法例におけるオペレータシーケンスのすべての任意の部分シーケンスをマクロオペレータの候補と考えてマクロを獲得したのでは，一つの解法例からでも多数のマクロオが生成され，さらに解法例が与えられる毎にマクロオペレータが著しく増加してしまう．よって，重複しない限りにおいてマクロの候補をすべて蓄える学習システムは，比較的少数の問題を解かせたときでさえマクロの増大により，問題解決の効率が学習無し問題解決システムよりも低下してしまうという大変興味深い現象がS.Mintonにより報告されている［Minton 85］．

このようにマクロオペレータの学習において，マクロの増大をどのようにおさえ学習無しシステムよりも高い効率を維持するかが最重要課題である．マクロの候補中には無意味なものが数多く含まれているのでそれらを取り除けば大幅にマクロを減少させることができる．しかし，数多くあるマクロの候補からどのように役に立つものだけを選択するかが問題となる．これに対する解決案としては［Minton 85］［Iba 85］などの方法があるが，ここで筆者は「完全因果性によるマクロオペレータの選択」という別のアプローチを提案する．

人間は問題解決においてマクロオペレータを用いていることは明かであるが，ではどのような基準でマクロオペレータを抽出しているのだろう

か？筆者はここで「完全因果性」というマクロオペレータの選択基準であるヒューリスティックを提案する．完全因果性は一次方程式の解法例において人間が自然に使っているマクロオペレータを構成しているオペレータ間にはどのような関係があるのかを分析することによって筆者が考えだしたものであり，以下のような仮説に基づいている．

1) 因果関係のないオペレータシーケンスはマクロオペレータにならない．

2) シーケンス中のあるオペレータシーケンスの適用結果が他のオペレータの適用を保証しているとき，それらのオペレータ間には強い因果関係があるとし，強い因果関係があるものだけがマクロオペレータになる．

2) の強い因果関係が筆者が「完全因果性」と呼んでいるものである．完全因果性は以下のように再帰的に定義される．

＜完全因果性＞

いま，OP1・・OPi・・OPj（1 ≦ i ≦ n：OPnは終了条件ルール）が解法例におけるオペレータのシーケンスとする．そして，OPi のそれぞれの具体化された条件部（拘束部も含む），結論部をPCi，Ai，またOPi以前で完全因果性のあるオペレータシーメンス：OPj・・OPkの結論部をAj・・Akとすると，

$$PCi \subseteq Aj \cup \cdots \cup Ak$$

が成り立つとき，OPj・・OPkとOPiには完全因果性があるという．このように完全因果性は一つのオペレータと一つのオペレータあるいはオペレータシーケンス間の性質である．具体例をFig.3.13に示す．この図では，PC2 ⊆ A1からOP1とOP2の間に，PC3 ⊆ A1 ∪ A2から {OP1,OP2} とOP3の間に完全因果性が成り立つ．

OPj・・OPkとOPi間に完全因果性があるということは，直感的にはOPj・・OPk が適用された後には必ずOPi が適用可能あるということを意味する．PiLは解法例におけるオペレータのシーケンスOP1・・OPnの各行OPi（1≦i＜n）から終了状態に向かって完全因果性を調べていく．そして，完全因果性が成り立つ最大のシーケンスだけを一つのマクロオペレータとして抽出する．完全因果性の定義はかなり難解であるが，マクロオペレータ抽出の具体的アルゴリズムは比較的簡単である．そのアルゴリズムをFig.3.14に示す．このアルゴリズムでは解法例中で連続していないオペレータシーケンスもマクロオペレータとして抽出可能である．

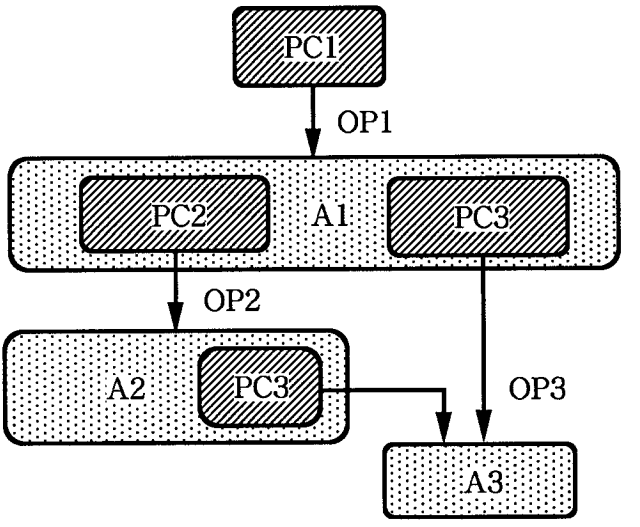


Fig.3.13 完全因果性

Fig.3.14のアルゴリズムによりFig.3. 8の解法例から求められたマクロオペレータの例が，Fig.3.15に示されたMO__1～6である．この図ではオペレータ間の因果関係がわかりやすいネットワーク表現がしてある．矢印が基本オペレータを表わし（OP1～OP5以外はラベル付けしていない），矢印の上付きの線，下付きの線がそれぞれ適用されたオペレータの条件部，結論部を表わす．例として，MO__5，MO__6の抽出過程について説明する．OP1～OP5は適用されたオペレータを表わし，それぞれの条件部，結論部をPC1～PC5，A1～A5とする．また，解法例中ではOP1～OP5の

順で適用されたとする.

まず, Fig.3.14のアルゴリズムで入力がOPS＝［OP1～OP5］でMOPS＝[], i＝1とセットされる. そして, MOP＝[OP1], RESULTはA1：(R3＊AL)／3＝R2／R3となり, OP2がA1に適用可能だから適用され, その結果の (R3／R3)＊AL＝R2／R3が新しいRESULTになり, MOP＝[OP1, OP2] となる. 次のOP3もRESULTに適用可能なのでまたRESULTが更新される. 以下同様にOP4, OP5も適用され, このループの結果はMOP＝［OP1,OP2,OP3,OP4,OP5］となり, MOP≠［OP1］なのでこのMOPがMOPSに加えられる. 次にOP2からのループが始める. まず, RESULT

```
INPUT (OPS) {OPS = [OP1 ･･･ OPn]}
MOPS← [ ]
i←1
WHILE i≠n DO BEGIN
   Let RESULT be the action−parts of OPi
   MOP← [OPi]
   j←i＋1
   WHILE j＜n DO BEGIN
      IF OPj is applicable in RESULT
         THEN ・RESULT←RESULT OPj applied
              ・assert OPj into MOP
      j←j＋1
   END
   IF MOP≠ [OPi]
      THEN assert MOP into MOPS
   i←i＋1
END
OUTPUT : MOPS is macro − operator list
```

Fig.3.14 マクロオペレータ抽出アルゴリズム

がA2：(R3／R3)＊ALにセットされ，OP3はこのRESULTには適用できず，次のOP4が適用され，RESULTが1＊ALとなる．このRESULTにOP5は適用できず，その結果MOP＝[OP2, OP4]となり，これがMOPSに加えられる．そして，OP3のループが始まるがRESUTLであるA3：R1

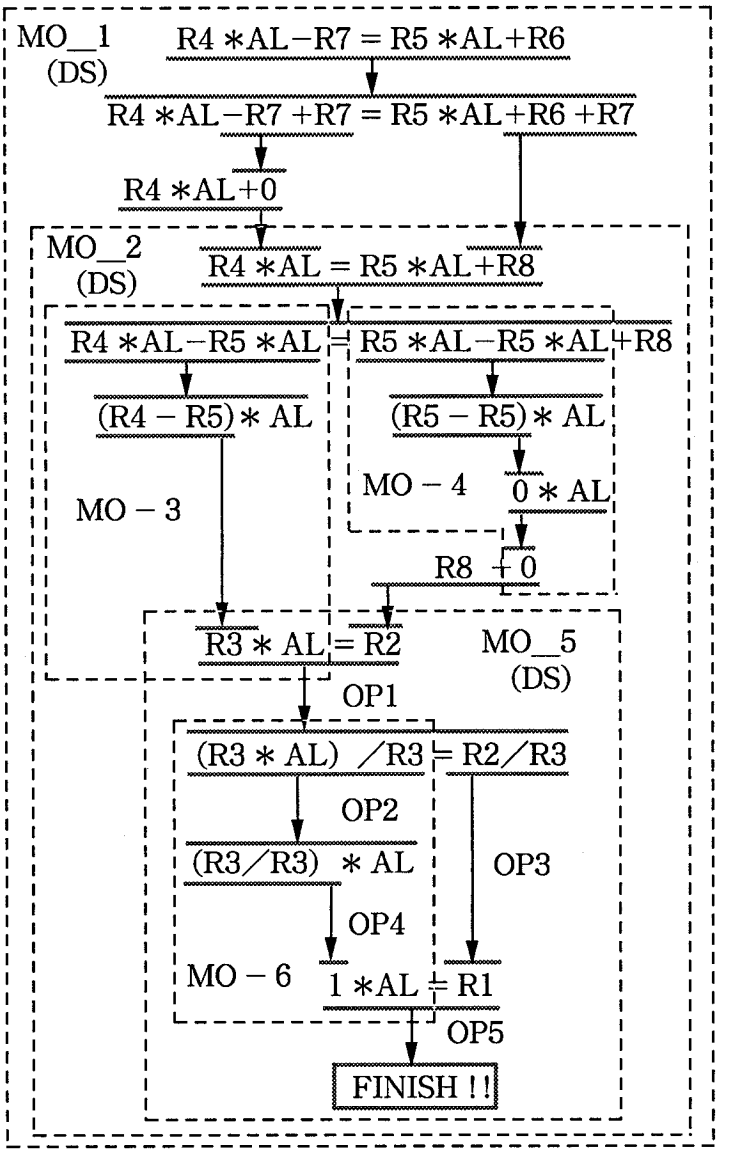


Fig.3.15 マクロオペレータの抽出

にはOP4, OP5は適用できず. MOP = [OP3] となり, マクロは得られない. 後は同様にOP4, OP5のループでもマクロは得られない. そして, 最終的な結果は, MOPS = [[OP1, OP2, OP3, OP4, OP5], [OP2, OP4]] となり, MO__5 : [OP1, OP2, OP3, OP4, OP5], MO__6 : [OP2, OP4] の二つのマクロオペレータが得られる.

終了条件まで含んでいるマクロオペレータ, つまりそれが適用されればワンステップで問題が解決されてしまうマクロオペレータを直接解決可能 (Directly Solvable) なマクロオペレータ, 略してDSマクロオペレータと呼ぶことにする. このDSマクロオペレータだけで解決可能な問題状態は, すべてのオペレータを用いて解決可能な問題状態よりも, 容易に判別できる. よって, DSマクロオペレータを用いて操作可能 (operational) なSOLVABLE概念 [Mitchell 83a,83b] を定義できるが, 詳細な議論は次章に譲る.

3. 5. 5 知識の整理

各オペレータの抽出が行われると, 知識の整理が行なわれる. PiLにおける知識の整理とは, 冗長な知識を削除することである. その結果, PiLの知識は非単調性を持つことになる. 知識の整理は, 1) 新しく得られたオペレータ内での整理, 2) 新しいオペレータと既存のオペレータ間での整理, の2つのステップがある. 絶対オペレータとヒューリスティックオペレータのそれぞれでは, この2ステップの整理がなされるが, マクロオペレータについては既存の知識と新しい知識が冗長な関係になることはないのでステップ1) のみがなされる. また, 整理の対象は同一種のオペレー

タである.

PiLにおいて対象としている冗長な知識には，全く同一の知識，より一般的な他の知識に包含される知識の2つがある．このような冗長な知識はステップ1)，2) とも以下の手順で同様に検出される.

いま知識の整理の対象としてるオペレータの集合の任意の二つの要素をOP1，OP2とする．以下の条件をすべて満たすとき，OP1は冗長な知識として知識ベースから削除される.

a) 構成オペレータの一致：OP1，OP2がヒューリスティックオペレータか絶対オペレータであり，かつOP1とOP2が同じ基本オペレータから生成されたものである．（この条件は冗長なマクロオペレータは原理的に生成されないことと，異なった基本オペレータから生成されたオペレータが冗長な関係にはならないことによる.）

b) 構造的一致：OP1とOP2の条件部が構造的に一致する．具体的には，まずオペレータの条件部の変数とアトム（演算子以外）の個数がそれぞれ等しく，かつ条件部が単一化できて，単一化の前後で変数とアトムの個数が変化しなければ構造的に等しいとする.

c) 概念関係：OP1がOP2と同一であるか，またはOP1はOP2の特殊な場合である．これはOP1，OP2のそれぞれの条件の拘束リストの概念述語の集合をR1，R2とすると前述の概念述語の記述の拘束から以下のような条件となる.

1) R1 = R2：このとき，OP1とOP2は同一のオペレータとなる.

2) R1がR2を含む．：このとき，OP1の適用範囲はOP2のそれに包含される．つまり，OP2はOP1よりも一般的なオペレータである.

この概念述語の集合間の包含関係はオペレータの変数をすべてアトム

に束縛し，条件部の単一化を行なった後に調べる．

具体例として，下に示す同じ基本オペレータから得られた2つのヒューリスティックオペレータについて考える．

OP1：(R1 * AL + R2)／NR1→R3 * AL + R4

条件部の拘束リスト：

[[alphabet, AL],[real__number, R1],[real__number, NR1],

[not__zero, NR1], [real__number, R2]]

結論部の拘束リスト：

[[equal, R3, [／, R1, NR1], [real__number, R3],

[equal, R4, [／, R2, NR1], [real__number, R4]]

OP2：(R1 * A + R2)／NR1→R3 * AL + R4

条件部の拘束リスト：

[[real__number, R1], [real__number, NR1],

[not__zero, NR1], [real__number, R2]]

結論部の拘束リスト：

[[equal, R3, [／, R1, NR1], [real__number, R3],

[equal, R4, [／, R2, NR1], [real__number, R4]]

この2つのヒューリスティックオペレータの違いは条件部の括弧内の最初の項がOP1では［任意の実数＊任意の変数］なのに対し，OP2では［任意の実数＊任意の整式］になっているところである．つまり，OP2の適用範囲がOP1のそれを包含するのでOP1は冗長である．このことを前述の手順で調べてみる．

まず，もとの基本オペレータは同じとしているのでa）構成オペレータの一致は成立ち，次のb）構造的一致を調べる．OP1とOP2の条件部の変

数の個数はともに3つでアトムの個数はともに0である．さらにこの2つの条件部は単一化可能であり，単一化後も変数，アトムの個数は変化しないので条件b）も満足される．そして条件c）を調べる．OP1，OP2それぞれの条件部の概念述語の集合R1，R2はR1：[[alphabet, a1]，[real_number, r1]，[real_number, nr1]，[not_zero, nr1]，[real_number, r2]]，R2：[[real_number, r1]，[real_number, nr1]，[not_zero, nr1]，[real_number, r2]]となる．（ここでは変数を例示化している）この2つの集合間にはR2がR1に含まれるという関係がある．この関係はR2の表わす概念はR1の概念を包含する，つまりより一般的であることを示すので，OP1がOP2の存在により冗長なヒューリスティックオペレータとなり取り除かれる．

3. 6 方程式解法の学習における実験

3. 6. 1 実験方法

ここでのPiLの対象分野は各種方程式の解法である．今回行なった実験で与えた問題は，1次方程式，1次不等式，2次方程式，分数方程式，対数方程式，指数方程式である．まず，Appendix1に示した基本オペレータ[Kasuga 75]をPiLに与え，次に各種問題のファイルを作っておき，それをPiLに読み込みせて解かせていく．そして，PiL自身で解けずに教師に解法例の教示を要請してきた場合に限り，教師がそれに答え解法例を教える．よって，PiLが独力で問題を解いている間は教師は何もしない．以降，このPiLに与える問題を「訓練例」と呼ぶことにする．

問題は中学生用あるいは高校生用の数学の問題集 [Bunmeido 85] [Bunken] [Sato 84] からほとんどそのまま抜粋し，さらにできるだけ幅広い問題を解けるようになるために足りないパターンの問題を若干追加した．ただし，分数・指数・対数方程式は適当な問題集がなかったので筆者が考えた比較的簡単な問題を各パターン1問づつ与えた．具体的にどのような訓練例を与えたのかをAppendix2に示す．これは今回の実験で与えた訓練例のすべてであり，その内訳は1次方程式：85問，2次方程式：211問，分数方程式：35問，対数方程式：78問，指数方程式：78問である．この問題のうち，1次方程式，2次方程式に関しては考えられるほとんどのパターンを網羅するようにしているが，分数・指数・対数方程式についてはPiLで扱えるごく基本的な問題だけに絞った．PiLで扱えない問題とは連立方程式や整式を変数で置き換えて解く問題（例えば，exp(9, x) + 2 * exp(3, x + 1) − exp(2, 4) = 0 でexp(3, x) = X とおいて解く）などがある．逆に扱える問題とは，Appendix2からも明らかなように1次・2次方程式のほとんどのパターンと比較的容易に1次・2次方程式に還元できるような分数・指数・対数方程式である．また，1次不等式も対応できる [Yamada 88a] がここでは正確なデータがないので割愛する．

訓練例を与える順序は方程式の種類でいうと1次方程式，2次方程式，分数方程式，対数方程式，指数方程式の順で与え，それぞれの種類における順序は問題集に載っている順序，つまり易しいものから難しいものへという順序で与えた．通常，例えばWinstonの学習システム [Winston 75] では訓練例を与える順序が学習の効率に著しく影響するし，またニアミスなどの有効な訓練例を選択して与えることが重要になる．これに対しPiLの学習能力においては「説明に基づく一般化」と「マクロオペレータ学

習」の両方が基本的に問題を与える順序に依存していないので，この易しいものから与えるという戦略は必然的なものではないことをつけ加えておく．

また，同一の問題でも複数個の解法がある場合がある．そのようなときはそれらのうちの一つだけを常に教えるようにした．例えば，2次方程式の解法では因数分解で解く方法は教えずに，完全平方に変形して解く方法を常に教えた．

3. 6. 2 実験結果とその評価

学習後，PiLは訓練例をすべて独力で解けるようになった．学習前は探索空間の爆発的拡大により訓練例のうち1問も独力では解くことができなかった．よって，学習後のPiLの問題解決の効率は学習前より飛躍的に良くなったことになる．また，当然訓練例以外でも獲得された戦略知識が使えるものであれば解決可能である．

Fig.3.16に1次方程式で得られたすべてのオペレータを示す，また，Appendix3に他の種類の方程式で学習実験後に得られたすべての戦略知識を示す．ただし，紙面の都合上，数の多いヒューリスティックオペレータは割愛した．さらに，各種方程式において学習された各種オペレータの個数をTable3.1に示す．

また，Appendix2の＃のついた問題はPiL自身では解けず教師に解法例の入力を要請した問題である．このような問題を「必須訓練例」と呼ぶことにする．もし，PiLに一般化能力がないなら，必須訓練例と訓練例は一致するはずであるが，一般化により必須訓練例が訓練例より少なくなる．

＜マクロオペレータ＞ 13

rule350 $ 0__31 [4] : (R1 * AL + R2) /NR1 → R3 * AL + R4
rule340 $ 8__32 [2] : (R1 * AL) /NR1 → R2 * AL
rule000 $ 5__30 [12] : R1 * AL + R2 = R3 * AL → 1 * AL = R4 (DS)
rule200 $ 0__29 [3] : (R1 * AL + R2) * R3 → NR1 * AL + R4
rule210 $ 16__27 [2] : A * R1 − A * R1 → 0
rule210 $ 33__28 [2] : AL * R1 + AL * R2 → NR1 * AL
rule010 $ 2__25 [15] : NR1 * AL + R1 = R2 * AL + R3 → 1 * AL = R4 (DS)
rule010 $ 11__26 [11] : R1 * AL = R2 * AL + R3 → 1 * AL = R4 (DS)
rule000 $ 3__24 [9] : R1 = NR1 * AL + R2 → 1 * AL = R3 (DS)
rule010 $ 2__23 [8] : NR1 * AL + R1 = R2 → 1 * AL = R3 (DS)
rule000 $ 3__22 [5] : R1 = NR1 * AL → 1 * AL = R2 (DS)
rule340 $ 6__3 [2] : (NA * AL) /NA → 1 * AL
rule030 $ 0__2 [4] : NR1 * AL = R1 → 1 * AL = R2 (DS)

＜絶対オペレータ＞ 6

rule310__16 : R1 * R2 → R3
rule230__24 : 0 * A → 0
rule300__22 : R1 + R2 → R3
rule270__5 : R1 − R1 → 0
rule320__21 : R1/NR1 → R2
rule280__13 : NA/NA → 1

＜ヒューリスティックオペレータ＞ 12

rule350__0 : (R1 * AL + R2) /NR1 → (R1 * AL) /NR1 + R2/NR1
rule000__5 : R2 * AL + R3 = R1 * AL → R1 * AL = R2 * AL + R3
rule200__0 : (R1 * AL + R2) * R3 → R1 * AL * R3 + R2 * R3
rule210__16 : B * R1 + B * A → (R1 + A) * B
rule010__11 : R1 * AL = R2 * AL + R3 → R1 * AL − R2 * AL = R2 * AL + R3 − R2 * AL
rule010__2 : NR1 * AL + R1 = R2 * AL + R3 → NR1 * AL + R1 − R1 = R2 * AL + R3 − R1
rule000__3 : R2 = NR1 * AL + R1 → NR1 * AL + R1 = R2
rule220__16 : 0 + NR1 * AL → NR1 * AL
rule010__2 : NR1 * AL + R1 = R2 → NR1 * AL + R1 − R1 = R2 − R1
rule000__3 : R1 = NR1 * AL → NR1 * AL = R1
rule340__6 : (A * AL) /A → (A/A) * AL
rule030__0 : NR1 * AL = R1 → (NR1 * AL) /NR1 = R1/NR1

Fig.3.16 1次方程式で得られるすべての戦略知識

	1次方程式	2次方程式	分数方程式	対数方程式	指数方程式
マクロオペレータ	13	62	58	56	57
絶対オペレータ	6	0	0	1	1
ヒューリスティックオペレータ	12	73	119	135	136

Table3.1 獲得されたオペレータの数

訓練例と必須訓練例との関係を示した表をTable3.2に示す．この与えた問題の総数と必須訓練例の差が，一般化によってその解法を教示しなくてもPiLが自力で解決可能な問題数を示している．よって，この差が大きいほど一般化能力が優れていると考えられなくもない．しかし，これは2つの別の学習システムの一般化能力の相対的な評価には使えるが，システムの絶対的な評価には使えない．

以上の実験結果をもとに以下にPiLの学習の学習能力について検討していく．

[1次・2次方程式の学習結果]

前述のように1・2次方程式についてはほぼ考えられるすべてのパターンの問題を与えたので，学習後のPiLはどのような1・2方程式でも解ける状態であると考えられる．そして，1次方程式についてはTable3.2からわかるように85問中のわずか8問について教師が解法を教えてやるだけで85問すべてが解ける状態，つまりほぼすべての1次方程式が解ける状態にまで達することがわかる．このことはPiLの学習能力，特に一般化能力の高さを示すものである．2次方程式においては211問中45問と必須訓練例は増えているが，これは2次式の方が式のパターンがかなり多いことによる．特筆すべきは，PiLは学習後にほぼすべての1・2次方程式を解

	1次方程式	2次方程式	分数方程式	対数方程式	指数方程式
訓練問題	85	211	35	78	78
必須訓練例	8	45	35	57	57

Table3.2 必須訓練例

答可能になっているので，これ以上いくら問題を与えても，すでに学習されている戦略知識で解決できるので，もう新しいオペレータは獲得されないということである．つまり，問題のパターンが有限個であればそのパターンをすべて学習し尽くせば戦略知識は定常状態に達するということになる．

Fig.3.16を見るとわかるように，変数項どうしの加減や分配法則による式の展開などPiLで得られたマクロオペレータは人が方程式を解く際に用いるマクロと類似していることがわかると思う．また，Fig.3.16のマクロオペレータのルール名の次の [] の中の数字は，そのマクロがいくつの基本オペレータで構成されているかを示している．多いものでは15個の基本オペレータから構成されているのがわかる．また，絶対オペレータとしては実数の演算を行なうオペレータなど，人間が優先的に用いている単独の基本オペレータが抽出されている．

Fig.3.17に定常状態に達したPiLが1次方程式を解く過程を示す．下線部は書き換えられる領域を示している．これを見ればマクロオペレータ，絶対オペレータの適用により問題状態が徐々にDSマクロオペレータの適用可能な状態に変化していき，最後にDSマクロにより一気に解かれるという解法過程が良く分かる．また，この問題をマクロオペレータを使わずに解くと26ステップかかるところが，マクロにより5ステップに軽減されている．このようにほとんどの問題はマクロオペレータと絶対オペレータだけで解ける．つまり，マクロオペレータと絶対オペレータをヒューリスティックオペレータよりも上位の階層におくことは問題解決の効率の点で非常に有効であることがわかる．

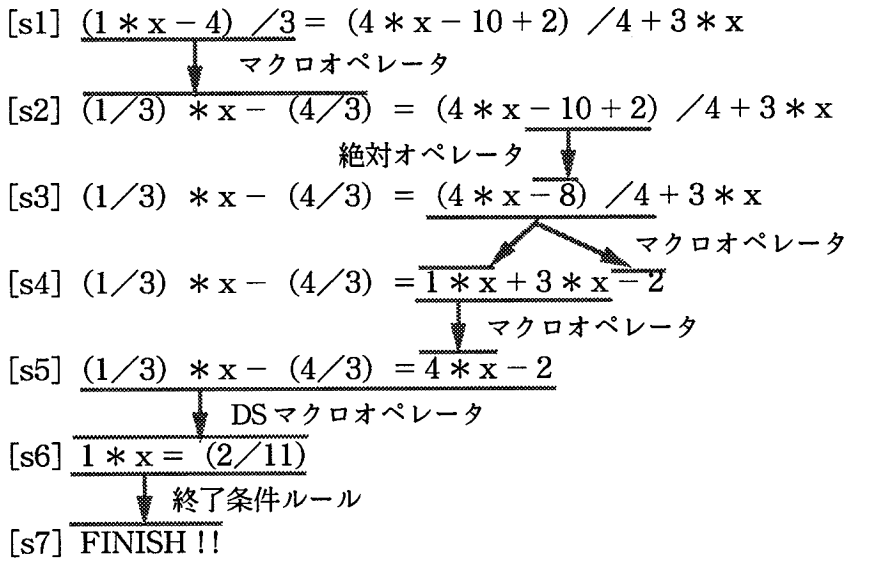


Fig.3.17 学習後のPiLの問題解決（1次方程式）

[分数・指数・対数方程式]

分数・指数・対数方程式においても同様の学習効果がみられる．これらの方程式では問題を解く場合のステップ数（適用される基本オペレータ数）が1・2次方程式よりも増えるのでそれにともないマクロオペレータ，特にDSマクロオペレータのステップ数が増え，マクロを使用することのメリットが大きくなる．Fig.3.18に学習後のPiLが対数方程式を解く過程を示すが，これは基本オペレータで解くと30ステップ以上かかるところがマクロオペレータによりわずか3ステップに軽減されており，その有効性を示している．

この実験結果で特徴的な点は，Table3.2からわかるように1・2次方程式の実験よりも必須訓練例が訓練例中に占める割合が大きいことである．これは問題集を参考にした1・2次方程式の実験では一つのパターン（例題）についていくつかの練習問題がついているのに対し，分数・指数・対

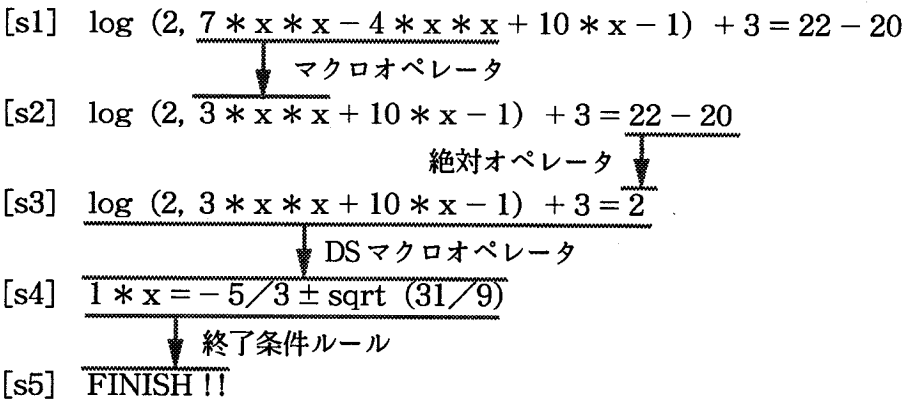


Fig.3.18 学習後のPiLの問題解決（対数方程式）

数方程式では参考になる問題集がなかったために，筆者が一つのパターンに付き一つの問題を与えたことに起因する．つまり，一つの問題を学習したことにより独力で解ける練習問題がほとんどないわけで，そのことが必須訓練例の割合の増加の要因になっている．

以上のように完全因果性によるマクロオペレータの抽出，絶対オペレータの生成というPiL独自の能力の有効性が実験的に証明された．

3. 7 関連研究との比較

1）T. M. MitchellらのLEX, LEX2

まず，学習システムとしてみたLEX [Mitchell 83a] とPiLとの主な相違点は以下の二点である．

a）LEXの一般化は類似に基づく一般化であるバージョン空間理論 [Mitchell 78] を用いているのに対し，PiLは説明に基づく一般化を使っている．

b) LEXはPiLのマクロオペレータに相当するものを全く学習できない. LEXが学習するものはPiLのヒューリスティックオペレータに対応する個々の基本オペレータの適用に関する戦略知識である.

a) についてはPiLの優れた点として, 一つの訓練例からでも学習が可能であり非常に学習効率がよいことと無根拠な一般化である「帰納的飛躍 (inductive leap)」を避け, 説明木による根拠のある一般化が可能である. さらに入力を比較すると, LEXは不定積分で使われる各種関数などの概念を記述する言語全体の階層関係を表わす概念ハイアラーキを入力として与える必要があるが, PiLでは基本オペレータの適用範囲に基づき一般化が行なわれるのでそのような全体的な概念ハイアラーキは必要としない. この全体的な概念ハイアラーキはシステム設計時に設計者が与えるが概念の関係を正確に設定することは難しく, またコンテクストにより変化する場合もあるので扱いが難しい. 反対に, LEXではこの基本オペレータに多少の誤りがあっても, 概念ハイアラーキが正確に記述されていれば一般化は正しく行なわれるが, PiLの場合は基本オペレータが正確に記述されないと正しい一般化が行なわれない.

また, b) についてはPiLは完全因果性により有効かつ基本オペレータに比べて少数のマクロだけを獲得できるので, 学習後のPiLはLEXよりもはるかに効率的に問題解決を行なうことができる.

LEX2 [Mitchell 83b,86] は一般化の効率を上げるために「説明に基づく一般化」を用いている. しかし, 基本的にはバージョン空間理論に基づいており, 類似に基づく一般化の限界を引きずっている. それに対して帰納的一般化をいっさい行なわないPiLの方が一般化の確かさにおいて優

れていると考える. また, マクロオペレータを学習できない点がLEX2でも改善されていないのでPiLの方が問題解決パフォーマンスにおいて優れていると思われる.

2) SilverのLP

LP [Silver 86a,86b] はPiLと同様に方程式の解法の学習を行なうシステムである. また, LPはスキーマといわれるPiLのマクロオペレータに対応する戦略知識を学習可能である. しかし, そのマクロオペレータの学習手法はPiLとは以下の点で異なる.

a) LPはスキーマ生成のための解法例の分割に既存のスキーマを用いる. よって, 最初にいくつかのスキーマを与えられている.

b) PiLのように等式を一般的なリスト表現だけで表わすのではなく, LPでは等式の特徴を表現する言語 : CT (Character Taple) [Furukawa 86] を最初に与えておき, それを用いてスキーマが記述される.

まず, a) の解法例の分割はPiLでは既存のマクロオペレータによって行なっているのではなく, 完全因果性を用いて分割しておりおり, 最初にまったくマクロオペレータがない状態でも当然解法例を分割してマクロオペレータを生成できる.

b) については学習システムの問題状態表現はできるだけ一般的でかつそれ自体に意図的な要素がないほうが望ましいので, 特徴を表わす言語をシステムに与えているLPの方が一般性に欠けると思われる.

3) PiLのEBG

PiLにおけるEBGをMitchellのEBG [Mitchell 86] と比較して考えて

みる．まず，PiLにおける説明に基づく一般化はMitchellらのEBGの枠組みに含まれるものであり，PiLの一般化アルゴリズム自体に独創性はない．ヒューリスティックオペレータの概念は［Mitchell 86］のusefu_opの概念と同様である．

以上のことからPiLシステムにおける独創性は，1）絶対オペレータの概念及びその抽出法，2）完全因果性というヒューリスティックによる有効なマクロオペレータの選択手法，の2点であると言える．また，R. Korf, S. Mintonなどのマクロオペレータに関する研究との比較は後の第5章で行なうことにする．

3. 8 問題点と今後の課題

1）無用なヒューリスティックオペレータの増加

実験結果から明らかなように，PiLで獲得されるヒューリスティックオペレータはかなりの数になる．そして，その中にはマクロオペレータに含まれておりそれ単独ではあまり役に立たないものが多く含まれている．そのため，2番目のマクロオペレータと絶対オペレータの階層だけで問題を解いているときは効率がよいが，その階層で解けずに下のヒューリスティックオペレータの階層へ探索が移った場合は多くの役に立たないヒューリスティックオペレータを評価したり，展開したりするために極端に効率が低下する．よって，無用なヒューリスティックオペレータを取り除くことが考えられるが，無用なものをどうやって見つけるかが明かでないため，現在のところ無用なヒューリスティックオペレータを自動的に排除する

ことは行なっていない.

3) 不完全あるいは誤った基本オペレータ

PiLにおいて基本オペレータがEBGの「領域知識」に対応する. EBGにおいて本質的な問題として領域知識が不完全な場合や誤っている場合にどう対処するかという問題があり, 当然PiLにおいても基本オペレータについてこの問題が生じる. ここではa) 基本オペレータが欠けている場合, b) 基本オペレータが誤っている場合について考える.

a) 基本オペレータが欠けている場合

このようなとき当然PiL自身で問題を解くことができず, 教師に解法例の教示を要請してくる. しかし, 目標状態にたどり着く途中の基本オペレータが欠けていれば, いくら教師の助力があっても解決できない. どの基本オペレータが足りないかをシステム自身が判断するのは難しいが, 教師はこうあるべき正しい解決過程を知っているので欠けている基本オペレータを比較的容易に検出できる. よって, 現在のところ教師がこの作業をしている. この検出過程を自動化するのが今後の課題の一つである.

b) 基本オペレータの記述が誤っている場合

この場合はa) よりもやや複雑である. 例えばある基本オペレータの条件部が必要以上に一般的に記述されている場合でも, 正しい解法例を得ることができ, その解法例生成の過程で教師が誤った基本オペレータを見つけるのは難しい. そして, その解法例が一般化され, そこから各種戦略知識が得られる. しかし, 基本オペレータが誤っているので得られた戦略知識は一般的過ぎるというように正しいものではなく, 次の問題を解くときにマクロオペレータの適切でない適用が起こり, そのとき初めて誤っ

た基本オペレータの存在がわかる．しかし，そのマクロオペレータを構成している基本オペレータから誤ったものを見つけ出すことは教師にとっても容易なことではない．さらに，もし見つけて修正したとしても，そのときまでに学習された戦略知識の内でその誤った基本オペレータを含んでものはすべて学習し直さなければならない．

4）誤った解法例

PiLでは最適で正しい解法例が得られることを前提にしている．PiL自身が問題を解決して得た解法例は横型探索を用いていることによりその最適性は保証されているが，教師の与える解法例は常に正しいとは限らない．もし最適でない解法例が与えられた場合，PiLは最適でない戦略知識を学習してしまう．しかし，教師によって与えられた解法例が最適かどうかをシステム自身が判断するのは困難であり，今後の課題である．

6）基本オペレータ中で演算子を変数にする

PiLのEBGにおいては基本オペレータをどれほど一般的に記述するかで，一般化の結果が違ってくる．そのため，基本オペレータは必要十分に一般的に記述される必要がある．今回，PiLが用いた基本オペレータをさらに一般的に記述する方法がある．それは基本オペレータ中の演算子を変数にし，，それにあわせて終了条件ルールの条件部の演算子も変数にすることである．具体的には例えば以下のようにする．

rule010 ： LHS OP RHS → LHS + A OP RHS + A,

[[member, OP, [=, <, >, =<,>=]]], [[not_zero, A]]

rule1000 ： 1 * A OP R → finish,

[[member, OP, [=, <, >, =<,>=]] [alphabet, A],

[real_number, R]], []

このように基本オペレータを変更することにより，1次方程式の解法を教えれば1次不等式も解けるようにすることが可能と考えられる．

7) 完全因果性及び絶対オペレータの普遍性

完全因果性によるマクロオペレータの選択及び絶対オペレータの抽出はPiLにおける独創的な部分であり，双方とも方程式の分野では有効であった．しかし，他の分野でははたして有効なのかという疑問が残る．この疑問については後の第5章でロボットの行動計画における実験で完全因果性の普遍性を検証する．

8) 問題状態表現の普遍性

7) において他の分野にPiLを応用することを考えると，問題状態の表現に問題がある．PiLは設計時に方程式の分野での応用を意識していたので，その結果問題状態表現及びすべての処理が数式のリスト表現に強く依存してしまい，汎用性を欠いてしまった．よって，リスト表現で表わせない分野に容易に適用できないという問題がある．問題状態表現としてもっと一般的な述語表現に対応できるように，システムを再構築する必要がある．これについても第5章で触れる．

9) ICAIへの応用

PiLで得られるマクロオペレータ，ヒューリスティックオペレータなどの戦略知識は正しい基本オペレータの使い方を表わしている．さらにこれ

らの戦略知識は教師が解いた解法例から得られるものなので（もちろん，システム自身が解く場合もあるが），教師がどのように問題解決においてオペレータを使っているかを表現しているとも言える．つまり，PiLで得られた戦略知識はICAIにおける教師モデルに相当する．教師は自分にとって容易に入力可能である問題の具体的な解法例をPiLに与えることにより，自動的に教師モデルを生成できることになる．このようにして得られた教師モデルをもとに生徒への教育を行なうことが考えられる．

3.9 まとめ

本章では，少数の解法例を与えられ，それを解析することにより問題解決における戦略知識を学習するシステムPiLについて報告した．そのなかで解法例の一般化，マクロオペレータ，絶対オペレータの生成などについて具体的方法を提案した．そして，各種方程式の解法学習により，PiLの学習能力の高さが検証された．なお，PiLシステムはSUN3上でK－PROLOGを用いてインプリメントされた．

第４章　直接解決可能性に基づく一般化：DSBG

近年,「説明に基づく一般化：EBG（Explanation-Based Generalization)」あるいは「説明に基づく学習：EBL（Explanation-Based Learning)」が機械学習におけるトピックスとなっており，様々な研究が活発に行われている [Rosenbloom 86] [Minton 87] [Mostow 87] [Salvik 87] [Hirsh 87] [Serge 87]．従来の帰納的一般化において問題であった静的な概念ハイアラーキによる無根拠な一般化である帰納的飛躍（inductive leap）[Mitchell 86] を避け，膨大な訓練例が必要なことによる学習効率の悪さを克服するものとして，事前に与えられた領域知識から演繹的な一般化を行うEBG（あるいは特殊化も行うEBL）が提案され，総括的な論文 [Mitchell 86] [DeJong 86] も発表されている．さらには，EBLとSBLとの融合を目指した研究 [Lebowitz 85] [Porter 85] もある．

本論文でも前章でEBGを用いた問題解決における戦略知識学習システム：PiL [Yamada 87b] を構築してきた．PiLシステムはprologを用いてのインプリメントが完了した後，実際に各種方程式の解法学習での実験が行われた．そして，この実験の過程でPiLのEBGにおける問題点が明らかになった．それは，PiLのEBGでは一般化がまだ不十分で，その過小一般化（under-generalization）により教師は冗長に思われる解法例を数多く教示しなければならないことである．一般にEBGは類似に基づく一般化に比べて一つの訓練例からでも説明木を構成することにより一般化が可能であり，同レベルの学習状態に達するまでに必要な訓練例の個数ははるかにEBGの方が少ない．しかし，方程式解法学習において訓練例である方程式の問題を与え，PiL自身が解けない問題の解法例を教示して行

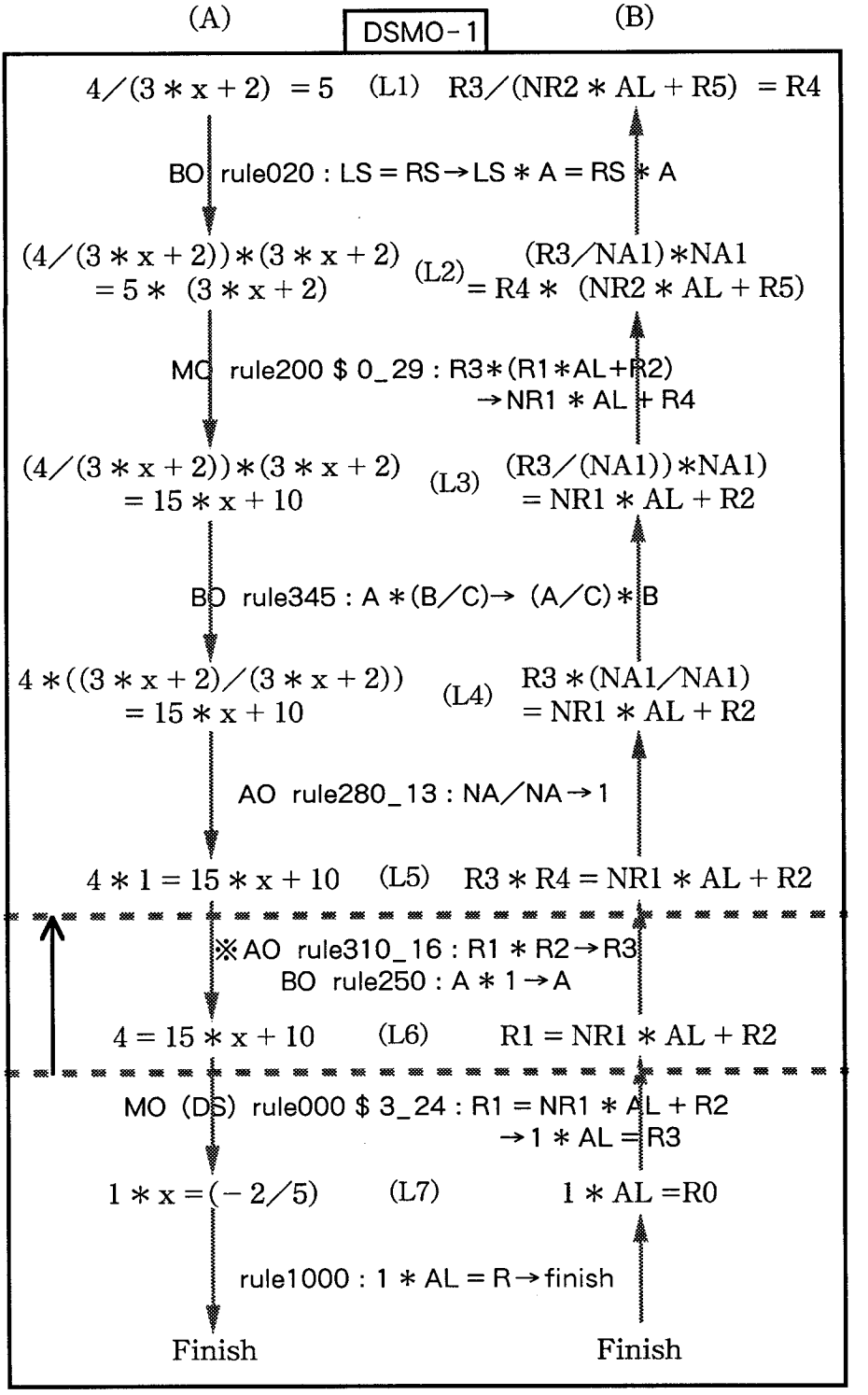
くと，教師から見ると冗長に見える問題の解法例を数多く要請してくることがわかった．筆者はこのEBGの過小一般化に対処するためにより十分な一般化を行なうには，Mitchellらの提案した解決可能な問題状態の内包であるSOLVABLE概念 [Mitchell 83a, 83b] を操作可能 (operaitonal) [Mitchell 86] [Keller 87a] な概念記述で再定義し，そしてEBGが操作可能なSOLVABLE概念を用いた一般化，つまりより短い説明木による一般化が可能なようにすればよいと考えた．その具体的手法として，本章においてSOLVABLE概念の操作可能なサブセットである直接解決可能概念：DS (Directly Solvable) 概念をDSマクロオペレータを用いて定義し，DS概念を用いてEBGを拡張した「直接解決可能性に基づく一般化：DSBG (Direct Solvability-Based Generalization)」[Yamada 88c, 88d] を提案する．そして，DSBGを実際にPiLシステムに適用して各種方程式の解法学習の実験を行い，従来のEBGを用いたPiLでの実験結果つまり前章での実験結果との比較・検討によりDSBGの有効性を検証する．

4. 1 PiLにおけるEBGの問題点

一般にEBGの問題点として，常に正しい領域知識と訓練例が与えられなければならない[Mitechll 86] ということがある．しかし，実際にPiLシステムに学習をさせていく過程で，もう一つの問題が生じた．それは，一般化が十分でないために，学習効率が悪い，つまり数多くの訓練例について解法例を教示しなければならないことである．EBGは従来の帰納的一般化に較べると，一つの訓練例からの一般化が可能であり非常に学習効率がよい．しかし，実際PiLシステムへの教示で，1次方程式からさらに2次

方程式，分数方程式，対数・指数方程式へとその解法が複雑になるに従ってPiL自身では解けずに教師が教示しなければならない解法例の量も増え，かつそれらの中には教師からみると冗長なものが非常に多く含まれていることがわかった．よって，教師の負担を軽減するために，より十分な一般化を行い学習の効率化を実現することが望まれる．

冗長な教示の具体例をFig.4.1で説明する．図中で（A）は解法例を，（B）はEBGによる一般化の過程を示している．図中のBO，MO，AOはそれぞれ基本オペレータ，マクロオペレータ，絶対オペレータを表わす．ここでは1次の分数方程式を教示しているが，この時点でPiLはあらゆるパターンの1次・2次方程式の解法をすでに教えられており，よってすべての1次・2次方程式を解決可能であるとする．この解法例から完全因果性によりFig.4.1の解法シーケンス全体が一つのDSマクロオペレータ：DSMO-1として抽出される．ここで（A）のL1の左辺の分子である'4'に注目して欲しい．（B）のようにEBGによる一般化が終了状態から伝搬されることにより，この'4'は（B）のL1に示すように任意の実数R3に一般化される．ところで，このマクロオペレータ：DSMO-1の本質的な操作は「両辺に左辺の分母と同じ整式を掛けて分母を払うことにより，分数方程式を1次・2次方程式に変形して解く」というものと考えられる．よって，この解法は本来はFig.4.2のような分数方程式（両辺に左辺の分母を掛けて1次・2次方程式に変形できるもの）に対しても共通して適用可能なはずである．しかし，マクロオペレータ：DSMO-1は，左辺の分子が「実数」にしか一般化されておらず，左辺の分子が実数でないFig.4.2のような問題には適用できない．よって，PiLはFig.4.1の解法例を教示しただけでは，Fig.4.2の問題は自力では解けず，その結果教師はFig.4.2の問



操作可能性の境界線

Fig.4.1 分数方程式の解法例

題一つ一つについて解法例の教示を行なわなければならない．このことは終了状態からのEBGでは一般化が不十分であることを表している．この様に冗長に思える解法シーケンスを多数教示することは教師にとって退屈かつ不快であり，Fig.4.1の一例を与えればFig. 4.2の様な場合全てに対してこのDSマクロオペレータを適用でき，自力で解決可能なように一般化することが望まれる．

$$\frac{3*x}{3*x+2}=5$$

$$\frac{3*x^2+6*x+1}{3*x+2}=5$$

$$\frac{3*x^2+1}{3*x+2}=5$$

$$\frac{3*x+4}{3*x+2}=5$$

$$\frac{3*x^2+6*x}{3*x+2}=5$$

$$\frac{3*x^2}{3*x+2}=5$$

Fig.4.2 冗長な訓練例

4. 2 直接解決可能性に基づく一般化：DSBG

前節で述べたPiLのEBGの問題点を操作可能性（operationality）[Mitchell 86] [Keller 87a] の観点から考えてみる．一般にEBGにおいて操作可能性の基準をどのようにとるか，具体的には説明木のどこで操作可能性の境界（bundary of operationality）を引くかによって，得られる目標概念の記述が違ってくる．直接解決可能性に基づく一般化：DSBGの鍵となる考え方は，マクロオペレータの一般化の説明木における操作可能性の境界を移動することによって，より十分な一般化を行なおうというものである．具体的には操作可能なSOLVABLE概念を定義し，それを用いて完全因果性で得られる説明木（基本オペレータシーケンス）の操作可能性の境界を上に移動させる．このSOLVABLE概念はMitchellら

によって提案された概念でオペレータの適用により最終的に解決可能である問題状態の集合である [Mitchell 83a,83b,86]. ところが, Mitchellらの SOLVABLE 概念の定義は操作可能なものではなかった. また, Kellerは META-LEX において SOLVABLE 概念の操作可能化を行なっている [Keller 87b] が, その方法は彼自身も認めているとおり非常に雑で基本的に生成-テスト法 (generate and test method) であり, 効率が悪い. さらに, Kellerの研究は操作可能な SOLVABLE 概念が EBG に与える影響についてまったく言及していない. そこで本章では領域に依存しない操作可能な SOLVABLE 概念の定義をより厳密に行ない, その SOLVABLE 概念を用いて従来の PiL におけるマクロオペレータの一般化の問題点を改善する新しい一般化手法である直接解決可能性に基づく一般化：DSBG (Direct Solvability-Based Generalization) を提案する.

4. 2. 1 直接解決可能性 (Direct Solvablity)

直接解決可能性とは, DSBG が用いる操作可能な SOLVABLE 概念であり, 次のように定義される.

定義：ある問題状態が DS マクロオペレータだけを用いて解決できるとき, その問題状態は直接解決可能 (directly solvable) であるという.

また, 直接解決可能な問題状態の集合を DS 概念といい, ある問題状態が DS 概念の外延であるか否かを判定する述語, つまり DS 概念の記述述語を述語 DS と呼ぶことにする. 述語 DS は1引数の述語で問題状態 (ここでは等式) を引数にとる. 具体的には述語 DS は引数である問題状態を現在蓄えられている DS マクロオペレータだけの適用で実際に解いて行き,

解決できれば真を返す．この探索は横型に行なわれ，ループだけがチェックされる．そして，適用されるDSマクロオペレータがまったく無くなったときに解決不可能として偽を返す．このDSマクロだけによる探索は，すべてのオペレータ（DSマクロ以外のマクロオペレータ，絶対オペレータ，ヒューリスティックオペレータ，基本オペレータ）を用いた探索よりもはるかに探索空間が小さく，よって相対的に検証が容易であり，その意味で操作可能であるといえる．DS概念はDSマクロオペレータのみで解決可能な問題状態なので，すべて解決可能な問題状態を表わしているのではなく，SOLVABLE概念の操作可能な部分集合であるといえる．解決途中にDSマクロオペレータ以外のオペレータを必要とする問題状態は，SOLVABLE概念には含まれるが，DS概念には含まれない．

また，DS概念はその時点で得られているDSマクロオペレータに依存するので，学習の過程で新たなDSマクロオペレータが得られるたびに動的に変化する．例えば，1次方程式だけを学習した状態ではDS概念は1次方程式のみをその要素としてもつが，さらに2次方程式を学習した後では1次・2次方程式を要素としてもつ．つまり，広い範囲の問題を学習して行くにつれて，得られるDSマクロオペレータも増え，それにともないDS概念も拡大して行くことになる．このDSマクロオペレータの増加にともない，述語DSの操作可能性が低下する危険もあるがそれについては4.5でふれる．

SOLVABLE概念の定義はこれまでにMitchell，Kellerらによって行なわれてきた［Mitchell 83a,83b,86］［Keller 87b］が，DS概念のようにマクロオペレータに基づいた定義は今までにはないまったく新しいものである．さらに，MitchellらのSOLVABLE概念は操作可能でなかったし，

Kellerのそれは非常に雑な操作可能化を行なっており，DS概念の方がより厳密で優れていると考えられる．

4. 2. 2 直接解決可能性に基づく一般化：DSBG

操作可能なDS概念が，マクロオペレータの生成にどのような影響を与えるかを考えていく．Fig.4.1ではこの解法例全体がDSマクロオペレータとして抽出された．このとき，PiLにとって操作可能な概念は終了条件ルールの条件部：1＊AL＝Rだけである．つまり，問題が解かれた状態の集合だけが操作可能な概念記述である．その結果，解法例の最も下のL6とL7の間に操作可能性の境界線が引かれる（Fig.4.1の破線）．この操作可能性の境界線は，操作可能な概念に含まれる問題状態と含まれない問題状態との間に引かれる．ところが，この解法例中のL6～L7でDSマクロオペレータが適用されている．DSマクロオペレータが適用されているということは，L6の問題状態がDS概念の要素であるということになる．DS概念は操作可能なので，操作可能性の境界線はL5とL6の間に引き上げられる．その結果，L6以下のオペレータシーケンスが切り取られ，残りのL1～L5でEBGによる一般化が行なわれる．このようにオペレータシーケンスが短くなることによって，より一般的に一般化されることになる．なぜなら，取り除かれたオペレータシーケンス中の拘束条件がまったく伝播されないからである．Fig.4.1の解法例中において操作可能性の境界線はDSマクロオペレータ：rule000 $ 3__24を越えて，ワンステップ移動するだけである．しかし，このDSマクロオペレータ中には多くの拘束条件が含まれているので，それが取り除かれたオペレータシーケンスを一般化することにより，Fig.4.1で得られるマクロオペレータよりもさ

らに一般的なマクロオペレータが得られるはずである.

このように操作可能なDS概念を用いて解法例中の操作可能性の境界線を移動させて一般化を行なう手法を「直接解決解決可能に基づく一般化：DSBG (Direct Solvability-Based Generalization)」と呼ぶことにする. 今までは, このDSBGを実現するために, まず解法例から完全因果性によって得られたDSマクロオペレータのうち他のDSマクロオペレータを含むものを探し, その含まれているDSマクロオペレータを削除して, 一般化を行なうという手続き的な手法を用いていた [Yamada 88c,88d]. しかし, 述語DSを使った領域知識をPiLシステムにつけ加えることにより, 宣言的で理解しやすいかたちでDSBGを実現できることがわかったのでここではその手法でDSBGについて説明して行く.

PiLシステムにつけ加える述語DSを用いた領域知識は, 以下のような終了条件ルールである.

rule1010： A = B → finish , [DS (A = B)], []

このルールにおいてA = Bは任意の問題状態（等式）と単一化可能であり, そのA = Bについての拘束条件がDS (A = B) である. つまり, このルールは「問題状態がDS概念の例であれば, 問題解決は終了する」という意味になる. このrule1010はPiLの終了条件ルールの入っている階層に選言 (disjunction) として付加されるが, 最もさきにこのルールの適用が調べられるようにしておく. またこのルールをつけ加えることはMitchellらのLEX2 [Mitchell 86] での領域知識の記述に従うと, そのSOLVABLE概念を次のように書き換えることに対応する.

$$\mathrm{SOLVABLE}(x) \Leftrightarrow (\exists\, op)\ (\mathrm{SOLVED}(op\ (x)) \vee \mathrm{SOLVABLE}(op\ (x)) \vee \mathrm{DS}(op\ (x)))$$

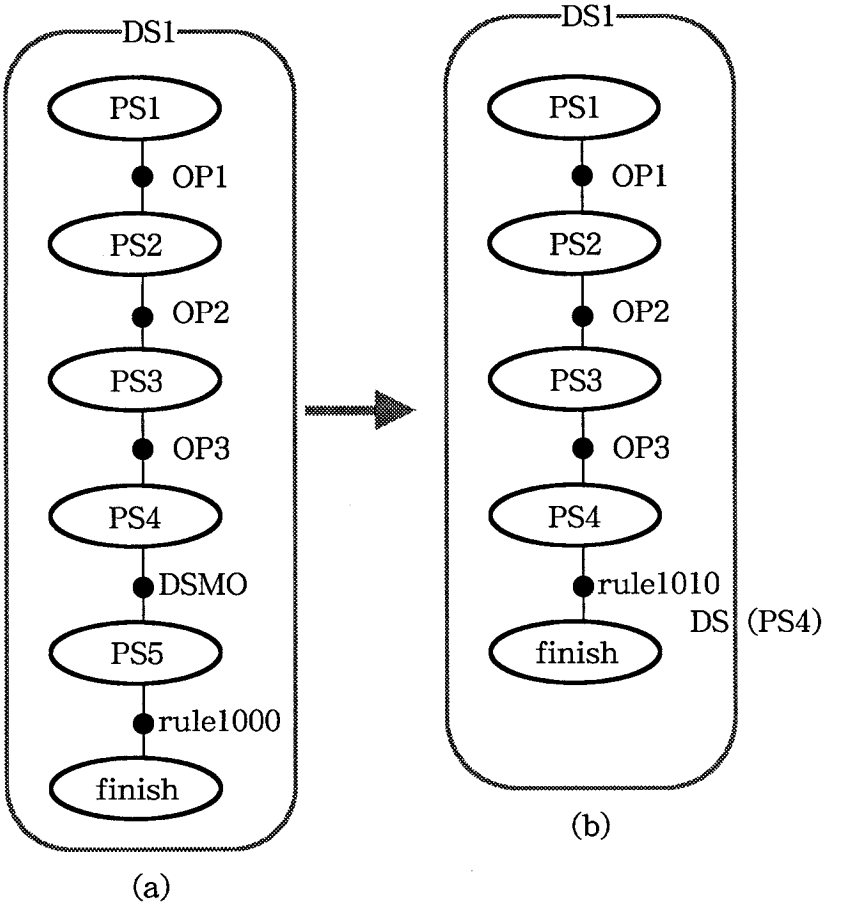


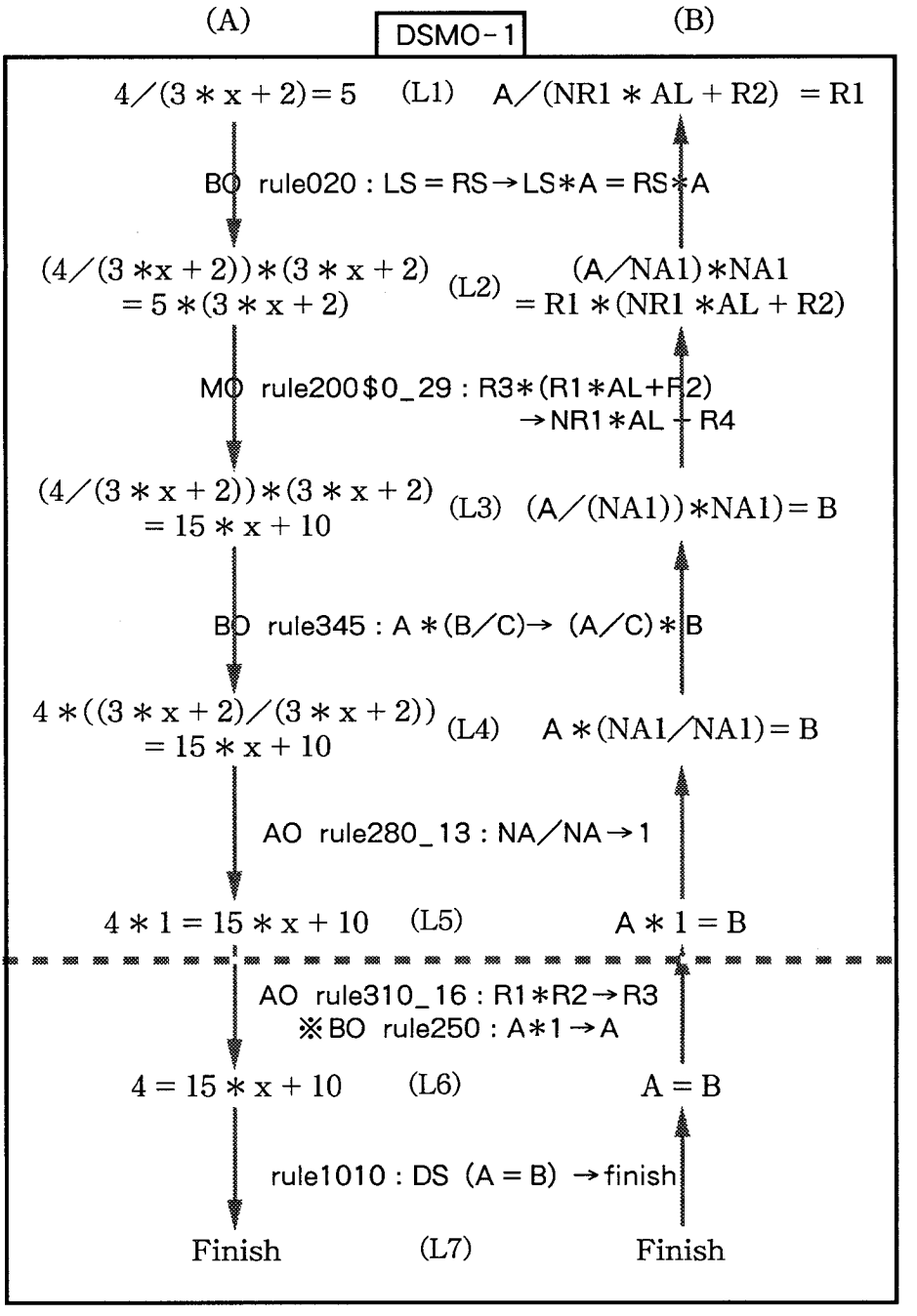
Fig.4.3 解法例(説明木)の変化

このような終了条件を付加することにより，得られる解法例がどのようには変わるかについて，Fig.4.3を例に考える．ここでFig.4.3（a）はrule1010が付加される前の解法例を表わしており，完全因果性により解法例全体が一つのDSマクロオペレータ：DS1として抽出されたとする．OPnは適用されたオペレータ，DSMOはDSマクロオペレータ，そしてPSnは問題状態を表わしている．（a）において，PS4にDSマクロオペレータが適用されているので，PS4がDS概念の要素である．よって，rule1010が付加された後の問題解決では，PS4にrule1010が優先的に適用されFig.

4.3（b）のような解法例が得られる．また，この（b）の解法例全体がDSマクロオペレータになっているので，そこからDSマクロが生成される．このDSマクロオペレータは拘束条件としてDS（A = B）を含んでいるが，（a）のDSMOが持っている拘束条件はまったく伝播されないので，（b）で得られるDSマクロオペレータは（a）で得られるものよりもより一般的なものとなる．また，DSBGで一般化できるマクロオペレータは，rule1010を含んでいなければならないので，DSマクロオペレータに限定される．

次にこうしてDSBGによって得られたDSマクロオペレータが，実際の問題解決時にはどのように適用されるのかを考える．DSBGによるDSマクロオペレータは必ず最後の問題状態をチェックする述語DSを拘束条件として持っている．例えば，Fig.4.3（b）から生成されたDSマクロオペレータは，OP1～OP3までの解法シーケンスを通過して得られた問題状態：PS4を述語DSでチェックする．そして，それが真なら問題は解決されたことになるし，偽ならそのDSマクロの適用は失敗し別のオペレータを適用する．このように述語DSによるチェックをしているので，過度に一般化されたオペレータを無駄に適用していたずらに探索空間を広げることを避けることができる．

具体的な例として，Fig.4.4にFig.4.1の分数方程式をDSBGで一般化していく過程を示す．Fig.4.4で（A）はrule1010を付加した後の解法例を表し，（B）が（A）を一般化した結果を表している．Fig.4.1においてL6でDSマクロオペレータが適用されていることからわかるように，L6はDS概念の例である．よって，Fig.4.4ではL6にrule1010が適用されて（A）のような解法例が得られ，これを一般化していく．この一般化



操作可能性の境界線

Fig.4.4　DSBGによる一般化

でL6からL5に伝搬する際，rule250とrule310_16の二つの伝搬経路が考えるので，最適経路の選択を行う．ここでは（B）のL6の変数A＝Bは任意の等式を表わすので，その左辺は任意の整式で良い．よって，結論部に任意の整式を持つrule250が最適経路として選択される．詳細は省くが目標状態からEBGにより一般化した場合は，逆にrule310_16の方が最適経路として選択される（Fig.4.1）．以下同様にL1まで一般化の伝搬が行われた結果がFig.4.4の（B）である．ここで注意すべき点は，3.5で述べたようにPiLのEGBはまず式を最大一般化した後，特殊化するということである．（B）のL1を見るとわかるように，EBGでは任意の実数R3に一般化されていた左辺の分子が任意の整式Aに一般化されている．これは最大限の一般化により任意の整式Aに一般化された左辺の分子を実数に特殊化する拘束条件が，Fig.4.1のDSマクロオペレータ：rule000＄3_24に含まれていたのでFig.4.4では伝搬しないからである．

こうしてFig.4.4から得られたDSBGによるDSマクロオペレータを以下に示す．

A／(NR1 ＊ AL ＋ R2)＝ R1 → 1 ＊ AL ＝ R4,

[DS （A ＝ NR2 ＊ AL ＋ R3)]

リスト中の述語DSは結論部の拘束条件で，その引数：A ＝ NR2 ＊ AL ＋ R3は条件部の等式にFig.4.4のL1からL6までの変形を行なった結果の等式である．つまり，このDSマクロオペレータの意味は「問題状態A／(NR1 ＊ AL ＋ R2)＝ R1をA ＝ NR2 ＊ AL ＋ R3まで変形し，その結果が直接解決可能であれば，この問題状態はワンステップで解ける」ということである．この時点で1次・2次方程式は学習済みなのでDSマクロオペレータの集合はFig.4.5のような状態である．Fig.4.2の問題をA ＝ NR2

＊AL＋R3に変形した結果は，すべてFig.4.5のいずれかのDSマクロペレータが適用可能であり，述語DSは真となる．つまり，Fig.4.1の一例を教示することにより，Fig.4.2のすべての問題が解決可能になる．その結果，教示しなければならない訓練例が減少し，学習の効率化が実現される．これは分数方程式の場合に限ったことではなく，指数方程式，対数方程式，無理方程式においてもDSBGにより学習効率が向上すると考えられる．また，重要なことはPiLはDSBGだけでなく従来のEBGでもDSマクロオペレータを一般化して残しておくことである．これはEBGで一般化されたDSマクロオペレータで解ける問題をDSBGによって得られるDSマクロオペレータで解く場合は述語DSのチェックを行なうのに計算コストがかかり，EBGのDSマクロオペレータで解く方が効率がよいからである．

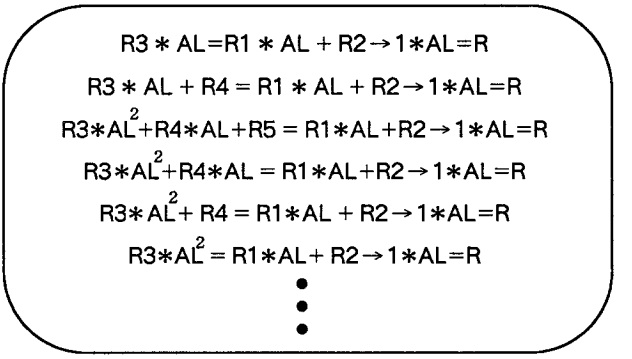


Fig.4.5 DS マクロオペレータ

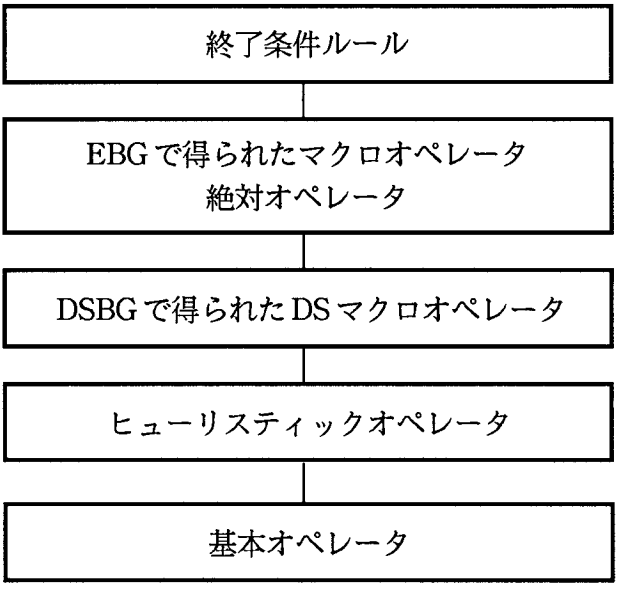


Fig.4.6 知識ベースの階層構造

次に，DSBGにより得られたDSマクロオペレータを知識ベースのどの

階層に入れるかであるが，これはFig.4.6のように絶対オペレータとEBGのマクロオペレータ（DSマクロを含む）の階層の下に新しい階層を作ってそこに入れる．また，DSBGのDSマクロオペレータで問題が解かれたときは，その解法例からEBGによってDSマクロオペレータを生成し，それを一つ上のEBGのDSマクロオペレータの入っている階層に入れることにする．以上のことによって問題解決の際に過去に解いたことのあるパターンの問題は上層のEBGによるDSマクロオペレータで効率的に解決され，過去に解いた問題と少し違ったパターンの問題はその一つ下のDSBGのDSマクロオペレータで解かれるという戦略が実現できる．

また，述語DSが問題状態の直接解決性を調べる際に，EBGのDSマクロオペレータとDSBGのDSマクロオペレータの両方を用いる．しかし，その優先順位は知識ベースのそれと同様で，EBGのDSマクロオペレータを優先する．

4. 3 各種方程式での学習実験によるEBGとの比較

4. 3. 1 学習能力の評価基準の選定

本節では，DSBGとEBGの学習能力を比較するための評価基準の選定を行う．ここで注意すべき点は一般化プロセスの改善により学習システムの能力の一部が向上しても，それに対してシステムの他の能力（例えば，問題解決能力）が低下し，システム全体の能力の改善になっていない場合が起こり得ることである．よって，できる限りシステム全体を総括的かつ本質的に評価できるような基準を選定することが重要である．一般に，問題解決における例題からの学習手法を評価する基準として，以下のものが

考えられる.

1) システムがあるレベルの学習状態に達するまでに必要な解法例の教示にかかるコスト.

2) 学習後の問題解決のパフォーマンスが学習前よりどれほど向上したか.

3) 学習プロセス自体にどれだけコストがかかるか.

この3つの評価基準のうち, ここでは1) と2) のみに着目するする. なぜなら, 3) は具体的にはCPUタイムなどで評価されるが, これは学習プロセスのインプリメンテーションをいかにうまく行うかに依存するので本質的な意味が希薄だからである. 次に, 1), 2) についてPiLに照らし合わせながら, さらに具体的に考えてみる.

まず, 1) であるが, 従来の評価基準では, この解法例教示のコストを定量的に考慮することはほとんどなかった. しかし, PiLのように解法例教示を人間が行うマン・マシン・インタラクティブな学習システムにおいては, この1) のコストは非常に重要である. また, この評価基準はハード・ソフトウェアの能力に依存せず, 学習システムの能力 (特に一般化能力) のみに依存するので本質的だと言える. PiLにおいては, 1) をある学習レベルに達するまでに, PiL自身が解けなくて教師が解法例を教示しなければならない訓練例つまり前章で述べた必須訓練例の個数で評価する. 「ある学習レベル」とは与えられた問題の集合がすべて解けるようなった状態を意味し, 具体的には与えた訓練例がすべてシステム独力で解けるようになった状態を表わす. 当然, この必須訓練例の個数が少ない方が優れた一般化能力を備えていることになる. また, ここでPiLが独力で問題を解けない状態とは, 展開されたノード (問題状態) 中に任意の整式 (変数) を含むものがある場合 (このとき展開は爆発する), または展開されたノー

ドの延べ数が100個以上になった場合であるとする.

次に, 2) の問題解決のパフォーマンス向上であるがこれは通常はCPUタイムで評価されることが多い. しかし, ここではより正確な評価を目指し, 探索空間の広さを表わす展開された探索木のノード数を調べる. 具体的には, 学習後のシステムに再び同じ訓練例を解かせた場合に解に至るまでに展開されたノード数を調べる.

4. 3. 2 訓練例教示と実験結果

今回は1次・2次方程式及び分数・対数・指数方程式の教示を行い, 前述の評価基準でEGBとDSBGを比較した. 前章の実験で用いた問題とまったく同じ問題を同じ順序で, 一般化の処理だけをDSBGに変えたPiLシステムに与えた. このことにより, 従来のEBGを用いたPiLの実験結果は3章の実験結果がそのまま使えることになる. そして, PiLが独力での問題解決が不可能なときに限り, 教師が解法例を教示する. このような教示法で, DSBGを用いたPiLは与えられた問題をすべて独力で解答可能な状態に至った. 今回の実験における必須訓練例とは与えた全ての問題を独力で解ける学習状態に達するまでに教師の解法教示が必要な訓練例 (方程式) を意味する. DSBGを用いたPiLも, 用いてないPiLもまったく同じ訓練例を学習後には独力で解けるようになったので, この2つのシステムは同じ学習状態に達したといえる. また, 問題解決のパフォーマンス向上の評価として, 探索木のノード数を調べたが, その結果前述の知識ベースの階層構造の設定により, EBGとDSBGでノード数は全く同じであった. 必須訓練例の結果をTable4.1に示す. また, DSBGとEBGとの必須訓練例の個数の比のグラフをFig.4.7に示し, Appendix4にDSBGの必須訓

	EBG	DSBG
1次方程式　ⓐ	8	6
2次方程式　ⓑ	45	30
分数方程式　ⓒ	35	5
対数方程式　ⓓ	57	3
指数方程式　ⓔ	57	3

Table4.1 必須訓練例

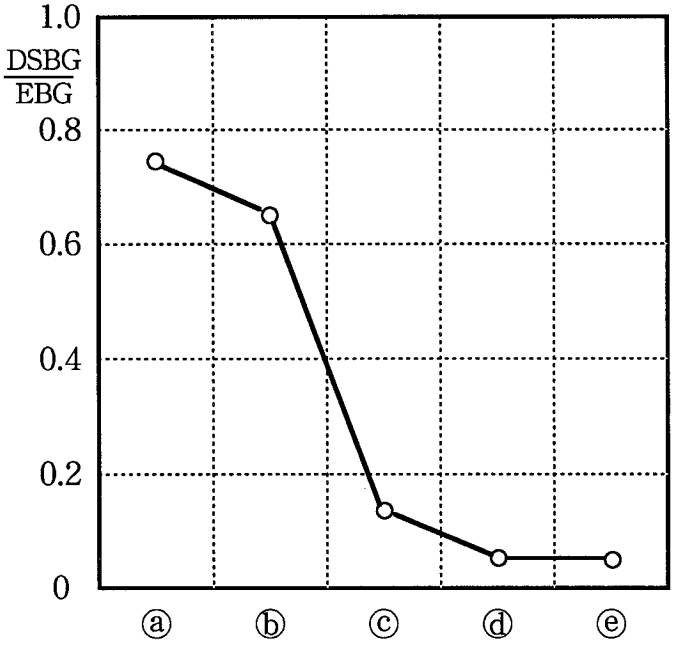


Fig.4.7 DSBGとEBGの必須訓練例

練例をすべて示す.

4. 3. 3 実験結果の評価

Table4.1, Fig.4.7の結果から明らかなようにDSBGではEBGよりも必須訓練例の個数が大幅に減少している. 特に, 分数・対数・指数方程式などの複雑な方程式でよりその傾向が顕著になっており, 対数・指数方程式におけるDSBGの必須訓練例の個数はEBGのそれの5%程度である. 教師がPiLに問題1問の解法例を教示するのにはかなりのコストがかかる（1問につき10分程度）ので, 必須訓練例の減少は非常に教師の労力軽減になる.

1次・2次方程式では与えた問題がほぼすべてのパターンの問題を網羅しているので, Table4.1の (a),(b) の結果は問題パターン全体にわたるものであるといえる. しかし, 分数・対数・指数方程式では与えた問題は問題のパターン全体からみると一部にしかずぎない. よって, Table4.1の結果は一部の問題パターンにおいてDSBGがEBGよりも優れているということで, 全体的には保証はないと言えるかも知れない. 確かに, Fig.4.7の必須訓練例の比が問題パターン全体で維持される保証はないが, Table4.1で減少している必須訓練例の個数は不変であり, 原理的にこれからさき種々のパターンの問題を与えたとしてもDSBGの必須訓練例がEBGのそれを上回ることは考えられないので, DSBGによる必須訓練例の減少は問題パターン全体からみても保証されていると考えられる.

また, 前述のようにノード数は変化していないので, 問題解決の効率は低下していない. これはDSBGにおいて, DSBGで得られたDSマクロオペレータで解かれた問題はEBGで一般化され, 上位の階層に蓄えられる

ので，EBGのPiLで解ける問題はDSBGを用いたPiLでもEBGで得られたDSマクロオペレータで解決できるからである．

よって，実験結果の総合的な評価として，DSBGは問題解決のパフォーマンス効率を損なうことなく，必須訓練例の教示コストを減少させて，システムの総合的な学習能力向上の実現に成功しているといえる．

また，Appendix5にDSBGで獲得された戦略知識のうち，マクロオペレータと絶対オペレータを示す．この中でルール名中にdsbgとつくDSマクロオペレータはDSBGのDSマクロオペレータで解かれた後にEBGで一般化されて得られたDSマクロオペレータである．DSBGではDSマクロオペレータがEBGよりもさらに一般化されて得られるので，DSBGで得られる一つのDSマクロオペレータがEBGの複数個のDSマクロオペレータに対応しているわけである．その関係をFig.4.8に示す．例1では1次方程式を解いたときに得られるNA＋R1＝R2→NA＝R3というDSBG

例1

< DSBG > $NA + R1 = R2 \rightarrow NA = R3$

< EBG >

$R1 * AL + R2 = R3 \rightarrow 1 * AL = R4$

$R1 * AL * AL + R2 = R3 \rightarrow A * AL = R4$

$\log(A, R1 * AL) + R2 = R3 \rightarrow 1 * AL = R4$

⋮

例2

< DSBG > $\log(A,B) = \log(A,C) \rightarrow B = C$

< EBG >

$\log(A, R1 * AL) = \log(A, R2) \rightarrow 1 * AL = R3$

$\log(A, R1 * AL + R2) = \log(A, R3 * AL) \rightarrow 1 * AL = R4$

$\log(A, R1 * AL * AL) = \log(A, R2 * AL) \rightarrow 1 * AL = R3$

⋮

Fig.4.8　DSBGとEBGのDSマクロオペレータ

のDSマクロオペレータは左辺の第1項が0以外の任意の整式でいいので，その下のEBGのDSマクロで表わされる2次方程式や対数方程式にも適用可能なことがわかる．また，例2ではDSBGのDSマクロオペレータはlogの第2引数が任意の整式でよいのに対し，対応するEBGのDSマクロではその整式が限定されており，これらのDSBGのDSマクロオペレータが必須訓練例の減少に貢献していることが良くわかる．

4. 4 関連研究との比較

1) Mitchellらの EBG

MitchellらのLEX2の研究［Mitchell 86,83b］でのSOLVABLE概念は文献［Mitchell 86,83b］［Silver 86a］でも述べられているように，ある訓練例がある概念のインスタンスであるのかどうかを容易に検証できないという意味で操作可能（operational）［Mitchell 86］［Keller 87a］ではなかった．EBGは「目標概念は操作可能な概念述語で記述されなければならない」という操作可能性の基準（Operationality Criterion）［Mitchell 86］に基づいて説明木を展開するで，LEX2ではPiLのEBGと同様に解法シーケンスの途中から一般化伝搬させることができず，問題が解決された最終状態（説明木の根）から一般化伝搬を行っていた．よって，得られた概念は当然必要以上に拘束が強いものとなり，その結果PiLでは冗長な必須訓練例が多数必要となったわけである．Mitchellらも領域に依存した操作可能なSOLVABLE概念（例えば，不定積分においてはあらゆる多項式の積分はsolvableである）を与えることについて述べている［Mitchell 83b］が，操作可能なSOLVABLE概念の一般的な定義を与える

ことはできていない．それに対してDSBGでは，まず「DS概念はDSマクロオペレータだけを用いて解決可能な問題状態の集合である」と定義する．「実際に探索してみて解ければ解決可能である」という検証法は本質的に正しいものであるが，すべてのオペレータを適用していたのでは，探索空間が広がりすぎて操作可能な概念にはならなし，またその検証過程自体が実際に問題解決を行うことと等価になる．DS概念の定義は，DSマクロオペレータのみを用いて探索を行うことにより，全てのオペレータを用いた探索よりも相対的に探索空間を縮小して，訓練例がある概念に含まれるかどうかの検証を容易にすること，つまりDS概念を操作可能にすることを意図している．そして後は対象分野における実際の学習過程で領域独立な基準である完全因果性に基づきDSマクロオペレータを獲得することにより操作可能なDS概念を自動的に生成することが可能である．ただし，DS概念は解決可能な問題状態全体を表すSOLVABLE概念と一致するものではなく，SOLVABLE概念の操作可能な部分集合であるといえる．

2）KellerのMETA-LEXにおけるSOLVABLE概念の操作可能化

第2章で詳述しているように，KellerのMETA-LEX［Keller 87b］も操作可能なSOLVABLE概念を自動生成できる．しかし，META-LEXではSOLVABLE概念を構成している述語をランダムに真か偽に入れ替え，そしてその後に実際に問題解決を行なって，パフォーマンスシステムの効率向上をチェックして操作可能なSOLVABLE概念を得るという方法を行なっている．しかし，Kellerの研究には，以下のような問題がある．

1）述語を真や偽に入れ換えた概念記述しか得られないので，表現力に乏しく，SOLVABLE概念の正確な記述を得ることは難しい．

2) 操作可能性を保証するために，実際にパフォーマンスシステムで問題解決を行ない，その有効性（utility）を検証していが，この検証には多大なコストがかかる．特に訓練例が多いときには膨大な計算コストがかかる．

3) Kellerの研究は操作可能なSOLVABLE概念がEBGにどのような影響を与えるのかについての考察がまったくない．

これに対して，DSBGのDS概念は概念記述が正確であり，意味論にも明確である．そして，DS概念の操作可能性は完全因果性というヒューリスティックで裏付けられているため，パフォーマンスシステムによる検証を必要とせず，その分META-LEXよりもコストが非常に軽減されるという優れた点がある．また，最も重要な点として，DS概念を用いることによってマクロオペレータ学習におけるEBGの過小一般化を克服できることを示し，さらに方程式の分野においてDSBGの有効性を確認したことが挙げられる．

4. 5 問題点と今後の課題

1) 不十分な一般化

Fig.4.4の解法例の一般化から明らかなように，DSBGでも，左辺の分母と右辺を任意の整式にまで一般化することはできない．これはFig.4.4のオペレータシーケンス中に，左辺の分母と右辺の式の構造を拘束する条件部をもったルールが含まれているからである（Fig.4.4のrule200 $0_29, rule020)．このことはDSBGでさえもまだ一般化が十分でないことを示しているともいえる．これをさらにDSBGの手法で一般化しようと

すると，DSマクロオペレータだけでなく，普通のマクロオペレータも用いて解ける問題状態の集合をDS概念と定義することになる．しかし，それでは述語DSはマクロオペレータすべてを用いて探索しなければならず，効率が悪化しDS概念の操作可能性が低下する．このような，一般化の十分性と操作可能性との間のtrade-off［Serge 87］に対し，どの地点でバランスをとるかは問題であるが，筆者はDSマクロオペレータによる直接解決可能性が妥当なものだと考えている．

2）DS概念の操作可能性の保証

DS概念の操作可能性が常に保証されるのかという問題がある．Kellerは以下の2つの条件を満たすとき，その概念記述は操作可能であるとしている［Keller 87a］．1）使用可能性（Usability：その概念記述が問題解決にとって使用可能である），2）有効性（Utility：その概念記述の使用により明記された目的について問題解決システムの効率が向上する）．従来の定義は「使用可能であれば有効である」という暗黙の前提があったのに対し，Kellerの定義は使用可能性と有効性を独立したものと考え，さらに有効性を「目的」に依存するものとしている点が非常に興味深い．このKellerの定義に従うとDS概念は使用可能性は常にみたしているが，有効性は必ずしも保証されていない．なぜなら，DSマクロオペレータが非常に増加した場合，DSマクロによる解決可能性の検証が他のオペレータを用いた場合よりも相対的に効率がよい，つまり有効であるとは限らないからである．しかし，操作可能性を実際にパフォーマンスシステムで検証しようとすると，KellerのMETA-LEXと同様にその検証にコストがかかってしまうというtrade-offがある．

3) 有害なマクロオペレータの生成

もう一つの重要な問題点として，全く不必要なマクロオペレータが獲得されることがある．今回の実験では下のマクロオペレータがそれで，2次方程式の教示の際に抽出される．

R1 * AL → 2 * R2 * AL

この役に立たないオペレータの発生は完全因果性に起因するものではなく，EBGの本質的な問題点である領域知識の不完全性によるものである．現在のところPiLはこのマクロオペレータによる無駄な探索の展開を人為的に防止しているが，これもシステムが自動的に検出・修正することが望まれ，今後の大きな課題である．

4) 複数の解法がある場合

今回の実験では，2次方程式の解法は完全平方に変形して解く方法だけを常に教示した．しかし，一般的に方程式の解法は一意に決まらない場合が多々あり，例えば2次方程式では他にも，因数分解して解く方法などが考えられる．もし，複数の解法を教師が教示した場合はどうなるだろうか．これは実験的に検証していないが，その場合はPiLは複数個の解法の戦略知識を獲得すると思われる．ただし，それらは問題解決の際に競合するので基本的に横型探索を行なうPiLはそれらをすべて展開していく．その結果いたずらに探索空間を広げることになるので，今回の実験では一つのパターンの問題の解法は1つだけに限定した．

5) 既存のDSマクロ

DSBGは解法例中に既存のDSマクロオペレータが適用可能な問題状態がある場合に可能である．しかし，既存のDSマクロオペレータが適用されなくても，そのいま得られている解法例から完全因果性によりDSマクロオペレータが得られる場合は，そのDSマクロオペレータを包含している別のDSマクロオペレータはDSBGの対象となる．つまり，その包含されているDSマクロオペレータの部分を述語DSで入れ換えることにより，DSBGが適用できるわけである．また，この場合には包含されるDSマクロオペレータが複数個ある場合が起こる（Fig.4.9 (a)）．この場合は最大限に一般化するために最も大きいDSマクロオペレータを取り除く（Fig. 4.9 (b)）．DSマクロオペレータ間は部分集合の関係しかありえないのでこの最大のDSマクロオペレータは一意に決まる．ただし，この手法はイ

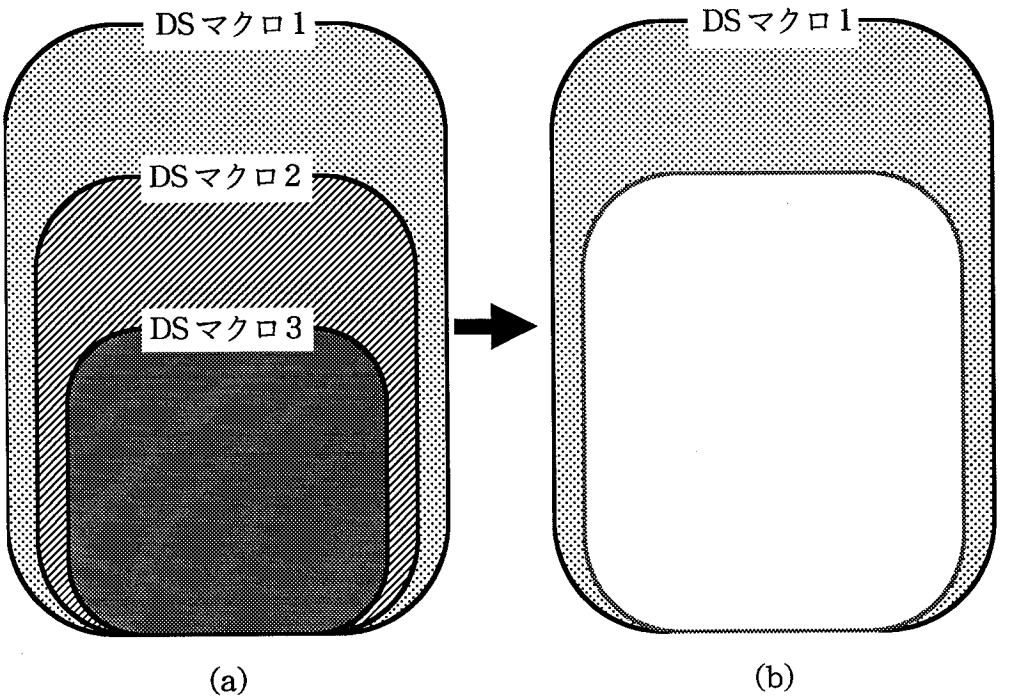


Fig.4.9 複数のDSマクロが包含される場合

ンプリメントはされていない.

6) 他分野での有効性

方程式以外の他の分野でもDSBGは有効かというのは非常に難しい問題である. DSBGの必須要素であるDS概念はDSマクロオペレータが理論的基盤になっているので, 原理的には有効かつ少数のDSマクロオペレータが得られる分野すべてにDSBGによる一般化は可能である. 次章で検証するように, 完全因果性によるマクロオペレータの選択的抽出は他の分野(ロボットの行動計画)でも有効なことがわかっているので, それにともないDSBGも普遍性をもつものと考えられる.

さらに, 現在のところPiLではDSマクロオペレータを含むすべてのマクロオペレータは完全因果性によって抽出される. しかし, DSBGはワンステップで解決状態に変形するDSマクロオペレータが生成されれば良いわけで, それがどのような方法(ヒューリスティックなど)で生成されるかは関係ない. つまり, DSBGは完全因果性に依存しないということである. 他の手法によってDSマクロオペレータが得られてもまったくかまわない.

4. 6 まとめ

従来のEBGを用いた学習システムでは, SOLVABLE概念が操作可能ではなく, そのために一般化が十分に行なわれず, 冗長な訓練例を多数教示しなければならなかった. この問題を解決するために, DSマクロオペレータを用いて操作可能なSOLVABLE概念の部分集合であるDS概念の

定義付けを行い，DS概念を用いてEBGを拡張した一般化手法である直接解決可能性に基づく一般化：DSBGを提案した．さらにPiLシステムにおいてDSBGを用いたものと従来のEBGを用いたものとで同じ訓練例を学習させる実験を行った．そして，その結果，DSBGにより大幅に必須訓練例の個数が減少することが確かめられた．

第5章　完全因果性によるマクロオペレータの選択的学習

3章において，問題解決における戦略知識の学習を行うPiLシステムの構築について述べた．PiLは問題状態を変化させる基本オペレータと問題解決の履歴である解法例を入力とし，「説明に基づく一般化：EBG」(Explanation-Based Generalization) [Mitchell 86] により解法例を一般化することで，個々の基本オペレータに関する戦略知識（ヒューリスティックオペレータ）と複数の基本オペレータを一つに合成したマクロオペレータの二種類を獲得可能である．このマクロオペレータは複数ステップを一度に実行することが可能で，効率向上が実現できる．しかし，通常一つの解法例からは多数のマクロの候補が考えられ，それらをすべて蓄えていたのではマクロが爆発的に増大してしまい，ついには学習無しシステムよりもパフォーマンスが低下してしまうことが報告されている [Minton 85]．このような問題に対処するには数多くのマクロの候補のうちから役に立つと思われるものだけを選択的に学習することが必須である．筆者はこのマクロオペレータの抽出基準として，3章において「完全因果性」というヒューリスティックを提案し，PiL上での各種方程式の解法学習においてその有効性を確認した [Yamada 88a, 88d]．

しかし，数式処理の分野で有効であった完全因果性がはたして他の分野にも適用できるのかという完全因果性の普遍性に疑問が残った．そこで本章ではより広範囲の問題領域に適用可能とするために過去の方程式解法での実験結果 [Yamada 88a,88d] を保証する範囲で完全因果性の定義を拡張し，さらにその普遍性を検証するために汎用マクロオペレータ学習システム：PiL2を構築して，ロボットの行動計画の分野における完全因果

性の有効性を検証する［Yamada 88e,89］.

まず最初に，PiL2の概要とその構成モジュールの働きについて述べ，そして拡張された完全因果性の定義，マクロオペレータ抽出アルゴリズムとEBGによる一般化手法を説明し，最後に実験方法・結果とその評価及び関連研究を示す.

5. 1 PiL2の概要

3章で述べたようにPiLシステムでは問題解決モジュール及びマクロオペレータを獲得・一般化する学習モジュールが数式のリスト表現に依存したものであった. しかし，リスト表現はかなり限定された表現形式なので，方程式解法以外の分野における完全因果性の有効性を検証するためには，PiLシステムの問題状態表現をより汎用性のあるものに変えなければならない. そこでより一般的な問題状態表現を扱えるようにと考えられたのが汎用マクロオペレータ学習システム：PiL2［Yamada 88e,89］である. PiL2のシステム構成をFig.5.1に示す. PiLにはないPiL2の特徴として，

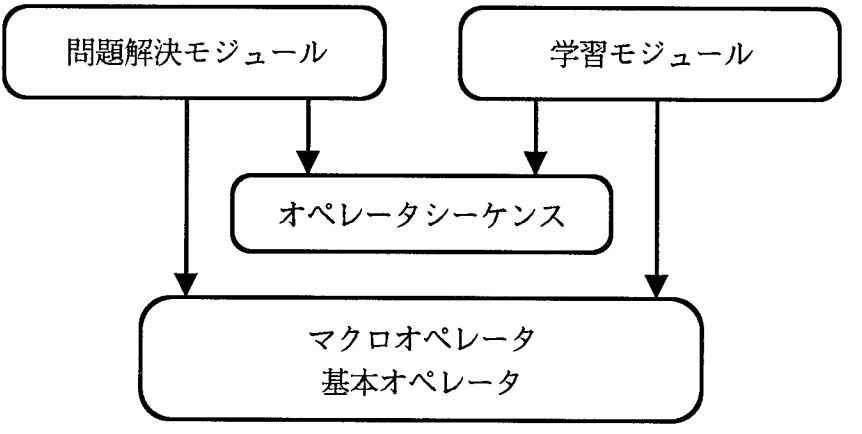


Fig.5.1 PiL2構成図

1）問題状態が述語表現である，2）拡張された完全因果性を用いる，3）戦略知識としてマクロオペレータだけを学習する，などがある．

以降，各モジュールの働きとマクロの選択的学習についてより詳細に述べていく．

5. 2 問題解決モジュール：STRIPS

PiL2は問題解決モジュールとしてロボットの行動計画で有名なSTRIPS [Fikes 71,72] を用いる．STRIPSのアルゴリズムは文献 [Fikes 71] に詳細に載っているので，ここでは要点だけを述べる．まず，世界の記述である問題状態は述語表現であるリテラルとルールで表わされ，その問題状態を変化させるのが基本オペレータである．STRIPSは問題の初期状態，目標状態及び基本オペレータを与えられ，その基本オペレータを使って問題状態を変化させ，目標状態を満足するようなオペレータシーケンス（ロボットの行動計画）を自動生成する．基本オペレータはFig.5.2のようなもので，条件リスト（cond-list），削除リスト（delete-list），追加リスト（add-list），主要効果リスト（main-effect-list）から構成されている．それぞれの意味は条件リストが満足されると現在の問題状態から削除リストと単一化可能なリテラルを取り除き，追加リストのリテラルをつけ加える．主要効果リストはそのオペレータにより得られる効果の主要なものであり，これが空リストの場合は主要効果リストが追加リストと同じであることを意味する．STRIPSは後向きに基本オペレータシーケンスをつくっていく．具体的には，現在できているオペレータシーケンスの先頭のオペレータの条件リスト中の述語で現在の問題状態で成り立っていないも

```
gotob (BX,RX) {Go to BX in RX}
   cond : [type (BX,object),inroom (BX,RX),
           inroom (robot,RX]
   delete : [nextto (robot,_)]
   add : [nextto (robot,BX)]
   main-effect : [ ]
pushb (BX,BY,RX) {Push BX to BY in RX}
        [type (BX,object),type (BY,object),
         pushable (BX),nextto (robot,BX),
         inroom (BX,RX),inroom (BY,RX)]
        [nextto (robot,_),nextto (BX,_),]
        [nextto (BX,BY),nextto (robot,BX)]
        [nextto (BX,BY)]
```

Fig.5.2 PiL2の基本オペレータ

のをサブゴールと考え，その述語を追加リスト中に持つオペレータ：関連オペレータ（relevant operator）をオペレータシーケンスの先頭に加えるという処理を再帰的に行なっていく．そして，目標状態が成り立つような基本オペレータシーケンスが得られたときに成功理に終了する．関連オペレータを探す際に，追加リスト中の主要効果リストだけが調べられ，効率を上げている．

また，STRIPSは関連オペレータが複数個ある場合にヒューリスティックを用いて最適なものを一つ選んで展開していく．PiL2で用いているノードの順序付けのヒューリスティックは，a）関連オペレータの適用により満たされるリテラル数，b）関連オペレータの適用可能性，c）繰り返されるサブゴールなどを評価するものである．

なお，STRIPSには2つのバージョン［Fikes 71,72］があり，一つは学習無しの問題解決システム［Fikes 71］であり，もう一つは学習可能な問

題解決システム［Fikes 72］でMACROPSという一般化されたマクロオペレータを獲得でき，それを三角表という表現で蓄え，後の問題解決に役立てることができる．

重要な点はPiL2は学習無しのSTRIPSを単なる問題解決モジュールとして用いていることである．また，PiL2のSTRIPSは基本オペレータとマクロオペレータの2種類のオペレータを別の階層に蓄え，まずマクロオペレータだけで探索して，行き詰まった場合に初めて基本オペレータの階層で探索を始める．

5. 3 マクロオペレータの選択的抽出とEBGによる一般化

nステップの解法シーケンスについてのマクロオペレータの候補数：MC（n）は順序関係を崩さない範囲でのすべての組合せであるからMC（n）$=\Sigma_{k=2}^{n}$nCkとなり，例えば10ステップだとマクロの候補はMC（10）＝1013にも昇る．つまり，マクロの候補は非常にたくさんあり，それらをすべて蓄えていたのではマクロが爆発的に増大してしまう．よって，学習有りSTRIPS［Fikes 72］のように重複しない限りにおいてマクロの候補をすべて蓄えるシステムでは，比較的少数の問題を解かせたときでさえマクロの増大により，問題解決の効率が学習無し問題解決システムよりも低下してしまうという大変興味深い現象がS.Mintonにより報告されている［Minton 85］．このようにマクロオペレータ学習において，マクロの増大をどのようにおさえ学習無しシステムよりも高い効率を維持するかが最重要課題である．ところで，マクロの候補の中には無意味なものが数多く含まれているのでそれらを取り除けば大幅にマクロを減少させることがで

きる．しかし，数多くあるマクロの候補からどのように役に立つものを選択するかが問題となる．これに対する解決案としては［Minton 85］［Iba 87］などの方法があるが，ここでは別の新しいアプローチを提案する．

5. 3. 1 拡張された完全因果性によるマクロオペレータの選択

人間は問題解決においてマクロオペレータを用いていることは明かであるが，ではどのような基準でマクロオペレータを抽出しているのだろうか？ 筆者は3章で「完全因果性」というマクロオペレータの選択基準であるヒューリスティックを提案した．しかし，残念ながら3章の完全因果性の定義では，拘束が強すぎて広い問題領域に適用することは難しいことがわかった．そこで本章では3章,4章での方程式の分野におけるマクロの学習結果を保証する範囲で，完全因果性の定義を拡張する．以下に拡張された完全因果性の定義を示す．

＜完全因果性＞

いま対象となる例示化されたオペレータシーケンスをOPS＝{OP1・・OPn}，問題の初期状態をISとする．ISはリテラルの集合であり，またオペレータはOP1・・OPnの順で適用されたとする．このときOPS中の完全因果性の成り立つオペレータシーケンス：PCOPS＝{OPi・・OPj}（1≦i＜j≦n）の任意の要素であるオペレータOPm（ただし，m≠i）は以下の条件を満たす．

OPmはISでは適用不可能であり，かつISに{OPi・・OPm-1}を順に適用した結果の問題状態では適用可能である．このとき{OPi・・OPm-1}とOPmには完全因果性があるという．

この条件は{OPi・・OPm-1}の適用がOPmの適用を保証しており，また

PCOPS中の任意のオペレータOPmと{OPi・・OPm-1} 間には必ず完全因果性が成り立つことを表わしている. 実際のマクロの抽出は {OP1・・OPn} の部分シーケンス：{OPk・・OPn} (1≦k≦(n-1)) それぞれについてOPkを含む完全因果性の成り立つ集合を調べ, そのうち最大のシーケンスだけをマクロオペレータとして抽出するということを行なう. その具体的なアルゴリズムをFig.5.3に示す. また, この完全因果性の定

```
INPUT (IS,OPS)   {IS is initial problem state,
                    OPS = [OP1 ・・・ OPn]}
Let IOP be OPS's operaotrs applicable in IS
MOPS← [ ]
i←1
WHILE i≠n DO BEGIN
   Let RESULT be the problem state after OPi
   was applied to IS without checking condition
   MOP← [OPi]
   j←i+1
   WHILE j<n DO BEGIN
      IF OPj ∉ IOP
        THEN IF OPj is applicable in RESULT
               THEN・RESULT←RESULT OPj
                                  applied
                    ・assert OPj into MOP
      j←j+1
   END
   IF MOP≠ [OPi] THEN assert MOP into MOPS
   i←i+1
END
OUTPUT : MOPS is macro − operator sequences
```

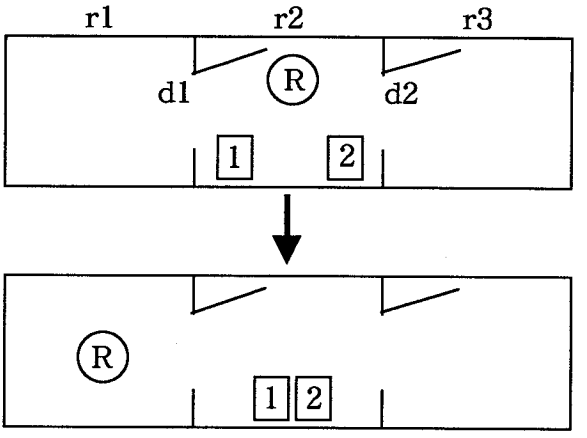
Fig.5.3 完全因果性による
マクロオペレータ抽出アルゴリズム

義は3章におけるものをより広範囲に対処できるように拡張したものであるが，3章と4章の各種方程式の実験結果を保証している．以上のマクロオペレータの抽出を以下に具体例で説明する．

例として，Fig.5.4のような問題とそれを解いたオペレータシーケンスが問題解決モジュールから与えられたとする．ここでOPS＝[OP1, OP2, OP3, OP4] であり，またこのシーケンス中の各リテラルはユニークなラベルづけ（fn）がされている．図中でf17とf19は同じリテラルであるが，それらを追加したオペレータ違うので区別している．

まず最初にIOP＝[OP1, OP3] が求まる．次にOP1が無条件にISに適用されその結果がRESULT＝[f0～f17] となり，MOP＝[OP1] とセットされる．次のOP2はIOPに含まれずかつその条件リスト：[f1, f2, f8, f11, f12, f17] がRESULTに含まれるので適用可能である．よって，OP2をRESULTに適用した結果である [f0～f16, f18, f19] でRESULTを更新し，MOP＝[OP1, OP2] になる．しかし，次のOP3はIOPの要素であり，またOP4はf20がRESULTにないので適用できず，以上でi＝1のループは終了し，MOP≠[OP1] よりMOP＝[OP1, OP2] が一つのマクロになる．

そして，i＝2で再びマクロの候補が調べられる．MOP＝[OP2] とセットされ，OP2が無条件にISに適用される．この際，削除リスト中のf17はISに存在しないので削除されず，追加リスト中のf18, f19が追加されRESULT＝[f0～f16, f18, f19] となる．次のOP3はIOPに含まれるのでとばしてOP4を調べる．しかし，OP4の条件リスト中のf20はOP3により追加されるものなので，OP4は適用できず，結局MOP＝[OP2] のままでこのループは終了する．そして，MOP≠[OPi] の条件が満足されず，



GOAL STATE : [nextto (box1,box2),inroom (robot,r1)]

IS = {f0 : type(robot,robot), f1 : type(box1,object), f2 : type(box2,object),
f3 : type(d1,door), f4 : type(d2,door), f5 : type(r1,room),
f6 : type(r2,room), f7 : type(r3,room),f8 : pushable(box1),
f9 : pushable(box2), f10 : inroom(robot,r2), f11 : inroom(box1,r2),
f12 : inroom(box2,r2), f13 : status(d1,open), f14 : status(d2,open),
f15 : connects(d1,r1,r2), f16 : connects(d2,r2,r3)}

OP1 : gotob(box1,room2)
cond : [f1,f10,f11] delete : [ ] add : [f17 : nextto(robot,box1)]
OP2 : pushb(box1,box2,room2)
cond : [f1,f2,f8,f11,f12,f17] delete : [f17]
add : [f18 : nextto(box1,box2), f19 : nextto(robot,box1)]
OP3 : gotod(door1,room2)
cond : [f3,f10,f15] delete : [f19] add : [f20 : nextto(robot,d1)]
OP4 : gothrudr(door1,room1,room2)
cond : [f3,f5,f10,f13,f15,f20] delte : [f10,f20]
add : [f21 : inroom(robot,r1)]
GOAL : [f18,f21]

Fig.5.4 問題とオペレータシーケンス

このループではマクロは得られない．以下，同様の処理が行われこの例から得られる最終的な出力はMOPS＝[[OP1：gotob，OP2：pushb]，[OP3：gotod，OP4：gothrudr]] となる．この二つのシーケンスはそれぞれ独立に行えるので，この結果は因果関係のない独立したオペレータシーケンスが完全因果性により切り離されて抽出されることを示している．また，このシーケンスのステップ数は4なのでMC（4）＝11ケのマクロの候補が考えられる．そのすべての候補をFig.5.5に示す．これらの中にはM3，M6などのように実行不可能なものやM4，M9のように意味のないものが多く含まれているが，完全因果性により11個から有効なもの2つだけが選択的に抽出されている．

また，このアルゴリズムを用いると連続なオペレータシーケンスから，非連続なオペレータシーケンスをマクロオペレータとして得ることも可能である．さらに，マクロオペレータと基本オペレータの混在したオペレー

M1：[gotob (OB1,R1),pushb (OB1,OB2,R1)]
M2：[gotod (D1,R1),gothrudr (D1,R2,R1)]
M3：[gotob (OB1,R1),gothrudr (D1,R2,R1)]
M4：[gotob (OB1,R1),gotod (D1,R1)]
M5：[pushb (OB1,OB2,R1),gotod (D1,R2)]
M6：[pushb (OB1,OB2,R1),gothrudr (D1,R3,R2)]
M7：[gotob (OB1,R1),pushb (OB1,OB2,R2),gotod (D1,R1)]
M8：[gotob (OB1,R1),pushb (OB1,OB2,R1),gothrudr (D1,R2,R1)]
M9：[gotob (OB1,R1),gotod (D1,R1),gothrudr (D1,R2,R1)]
M10：[pushb (OB1,OB2,R1),gotod (D1,R2),gothrudr (D1,R3,R2)]
M11：[gotob (OB1,R1),pushb (OB1,OB2,R1),gotod (D1,R1),
gothrudr (D1,R2,R1]

Fig.5.5 マクロオペレータのすべての候補

タシーケンスやマクロオペレータだけで構成されたオペレータシーケンスに対しても適用可能である.

5. 3. 2 もう一つの拘束条件

完全因果性によるマクロオペレータの選択によって因果関係の希薄なマクロの生成が抑えられるが, それでもまだ有効とは思われないマクロが生成されることがある. それは複数のマクロによって一つのマクロが生成されることである. 例えば, 2つ隣りの部屋へ行くときにmacro (gotod, gothroudr), macro (gotod, gothroudr) のようなオペレータシーケンスが得られた場合, 完全因果性によりこのシーケンス全体が一つの新しいマクロとして生成される. もしこのような既にあるマクロで構成されるマクロを蓄えていくとどうなるだろう. マクロの階層には個々のマクロとそのマクロを複数個組み合わせたマクロが混在することになる. これは基本オペレータとその組合せであるマクロが混在している状態と等しく, そのような状態では効率は向上しない場合が生じる. よって, 《現存のマクロによって構成できるマクロは生成しない》という拘束条件をつける.

5. 3. 3 説明に基づくマクロオペレータの一般化とその合成

以上のようにして得られたマクロの候補であるオペレータシーケンスはインスタンスなのでこれを一般化しなければいけない. ここでは「説明に基づく一般化：EBG」の手法 [Mitchell 86] で一般化を行なう. 既に, 3章でマクロオペレータの一般化・合成は行っているが, 述語表現によってそれらの処理がより明確に記述できるので, ここで改めて説明していくことにする. 一般にEBGでは次の4つの要素が必要である.

1）目標概念（Goal Concept）

2）訓練例（Training Example）

3）領域知識（Domain Theory）

4）操作可能性の基準（Operationality Criterion）

マクロオペレータの一般化で上述のそれぞれの要素が何に対応するかを考える．まず，前節のようにして抽出されたオペレータシーケンスはEBGにおける「説明木」にあたる．例としてFig.5.4から抽出されたオペレータシーケンス：[gotod, gothrudr] によって構成される説明木をFig. 5.6に示す．この図では黒丸がオペレータを表わし，上，右，下についているノードがそれぞれ追加，削除，条件リストを表わす．また，点線のノード

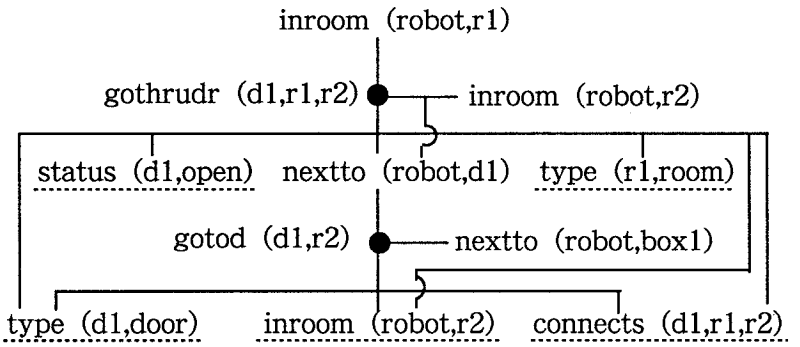


Fig.5.6 説明木

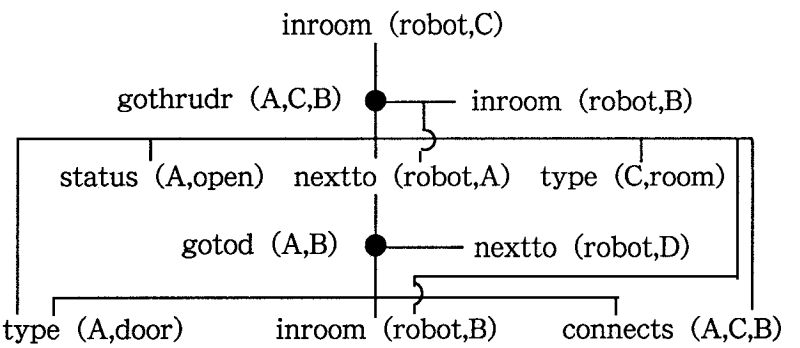


Fig.5.7 一般化された説明木

が訓練例に，それから基本オペレータgotod，gothrudrが領域知識に対応する．また，操作可能性の基準は完全因果性に対応すると考えられ，目標概念はLEX2［Mitchell 86］において，あるオペレータの適用が有効なときの条件を導くUSEFUL-OP（COND）と同様なものでこの例の場合は例えばUSEFUL-MACRO（COND）とかける．

一般化過程の詳細は割愛するがMitchellらのEBG［Mitchell 86］によりFig.5.6を一般化した結果がFig.5.7である．次にこの一般化された説明木から条件リスト，削除リスト，追加リスト，主要効果リストを抽出して一つのマクロを生成する方法をFig.5.8で説明する．この図は一般化された二つのオペレータのシーケンスを表わし，Cn，Dn，An，MEnはそれぞれ条件リスト，削除リスト，追加リスト，主要効果リストを表わす．各オペレータの各リストは述語の集合と考え，以下のような集合演算を行なうことにより新しいマクロの各リストが生成される．MC，MD，MA，MME

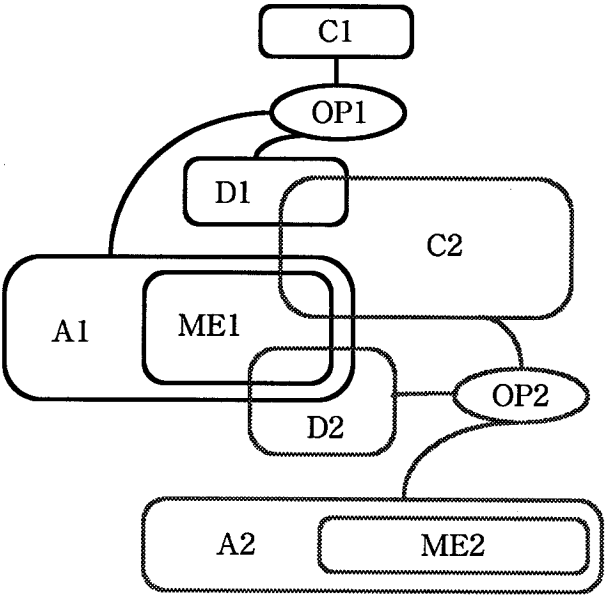


Fig.5.8 マクロオペレータの合成

はそれぞれ合成されたマクロの条件リスト，削除リスト，追加リスト，主要効果リストであり，上付きの線は補集合を表わす．

$MC = C1 \cup (C2 \cap \overline{D1} \cap \overline{A1})$

$MD = (D1 \cup D2) \cap \overline{A1}$

$MA = A2 \cup (A1 \cap \overline{D2})$

$MME = ME2 \cup (ME1 \cap \overline{D2} \cap \overline{C2})$

この手順をマクロの候補であるオペレータシーケンスに再帰的に適用することにより基本オペレータが合成されマクロが生成される．例えばFig. 5.7からは以下のマクロオペレータが得られる．

macro (gotod (A, B), gothrudr (A, C, B))

MC : [type (A, door), status (A, open),
type (C, room), inroom (robot, B),
connects (A, C, B)]

MD : [nextto (robot, _), inroom (robot, B)]

MA : [inroom (robot, C)]

MME : [inroom (robot, C)]

5. 4 実験方法

以上のマクロの選択的学習がロボットの行動計画において有効なことを示すためには，どのような実験を行なえばよいのだろうか．ここでは客観的に妥当なものと考えられるMinton [Minton 85], Fikes [Fikes 72]の実験方法を用いることにする．

まず，実験に用いるシステムはつぎの3つである．

(a) 学習無し問題解決システム：STRIPS

(b) 選択的でないマクロオペレータ学習システム：M-STRIPS

(c) 選択的マクロオペレータ学習システム：PiL2

(a) はPiL2の問題解決モジュールであるSTRIPSをそのまま用いることにより実現できる. また (b) はPiL2において全く選択せずにすべての候補からマクロを生成することにより実現できる. ここで重要なことは, この3つのシステムの問題解決モジュールは全く同じものでその違いは獲得されるマクロの質と量だけということである.

つぎに問題解決のパフォーマンスを何をもって評価するかであるがここでもFikes [Fikes 72], Minton [Minton 85] などで使われており一般的と考えられる以下のものを採用する.

1) 評価された枝の数 (Branches evaluated)・・・解決に至るまでに評価された枝の合計. 展開されてないものも含む.

2) 展開されたノード数 (Nodes expanded)・・・解決に至るまでに展開されたノードの合計.

3) 解決ステップ数 (Solution length)・・・解決経路のステップ数.

4) マクロオペレータの数・・・その問題を解いているときのマクロの数.

5) CPUタイム・・・解決に至るまでのCPUタイム (秒)

次にどのような問題と基本オペレータで実験を行なうかであるが, 基本オペレータはSTRIPSの論文 [Fikes 72] に載っている6つのオペレータ (Appendix6) をそのまま抜粋して用いた. これはPiL2にとって都合のよい基本オペレータで実験したのではないことを強調するためである. また, 与えた問題は全部で50問で, STRIPSが解決可能なものである. そして, それらをやさしいものから徐々に難しいものへという順序で与えた.

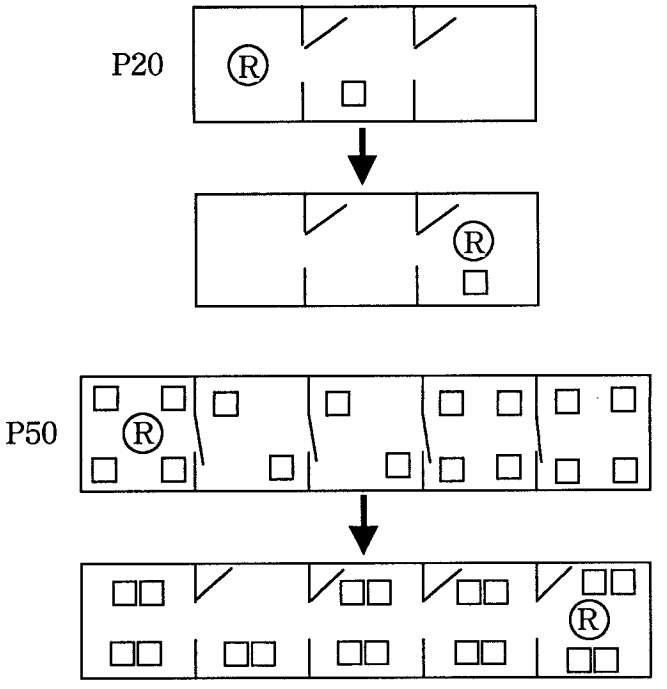


Fig.5.9 PiL2に与えた問題の抜粋

Appendix7に今回与えたすべての問題を，Fig.5.9 にその抜粋を示す．なお，与えた問題をSTRIPSで解いた場合の解決経路の最大ステップ数は28である．

5. 5 実験結果とその評価

Table5.1に50問中の5問を抜粋した実験結果を，Appendix8～11にすべての結果を示す．これらの実験結果において，P14以降はM-STRIPSの結果がなくSTRIPSとPiL2の結果だけであるが，これは残念なことにM-STRIPSがP13を解いて，そのオペレータシーケンスからマクロを生成するところでスタックがオーバーフローしてしまいそれ以上実験を続

けられなかったからである. なお, P1～12でM-STRIPSが生成したマクロ数は202個, P13を解いたあとでそのシーケンスから生成しようとしたマクロの候補は502個であった. また, M-STRIPSはこの202個のマクロを一通り探索するのにCPUタイムで約200秒かかるのでP14以降は必ず200秒以上かかることになる. 以下, Table5.1を中心に実験結果について検討していく.

＜P1～13＞Table5.1のP10のCPUタイムを見ると, すでにこの時点でM-STRIPSはSTRIPSよりも効率が低下している. 評価された枝はSTRIPSよりも少ないのだが, 一般にマクロの条件リストは基本オペレータよりも長く, 従って同じ個数のオペレータを探索するとマクロの方が時間がかかる [Minton 85] ので, その影響が出ていると考えられる. またすでにP10でM-STRIPSは70ものマクロを持ってしまっている. 対照的にPiL2は

problemID	P10	P20	P30	P40	P50
Branches evaluated	180 : 140 : 9	72 : 10	315 : 20	486 : 20	756 : 60
Nodes expanded	31 : 9 : 5	16 : 5	51 : 7	77 : 9	155 : 54
Solution length	8 : 2 : 3	5 : 2	10 : 4	12 : 4	28 : 12
Macro－operators	0 : 70 : 3	0 : 5	0 : 5	0 : 5	0 : 5
CPU time (sec)	47 : 104 : 10	20 : 12	114 : 19	244 : 27	1819 : 197

(P10) STRIPS : M-STRIPS : PiL2　(P20～P50) STRIPS : PiL2

Table5.1 Experimental results

評価された枝もマクロも非常に少なく当然効率も最も良い.

＜P14～50＞P14以降はPiL2とSTRIPSとの比較になる. まず最も特徴的なことはPiL2のマクロが最後の訓練例であるP50のときでさえたった5個しかないことである. その5個のマクロをFig.5.10に示す. これらのうち, M5はM4に包含されるが, M4とM5の差が1つの基本オペレータ：gotodであり, マクロではないので5.3.2の拘束条件には違反しない. マクロをもし選択的に学習していなければ少なくとも1000個以上にはなると考えられるので完全因果性により著しくマクロの生成が抑えられていることがわかる. また, この少数のマクロオペレータが役に立つものであることの証明としてPiL2の解決ステップ数がSTRIPSの半分以下に減少しており, 当然それにともなって評価された枝数, 展開されたノード数

```
M1 : [gotod (D,R1), open (D),
      gothrudr (D,R2,R1)]
M2 : [gotod (D,R1), gothrudr (D,R2,R1)]
M3 : [gotob (A,R1), pushb (A,B,R1)]
M4 : [gotob (A,R1), pushd (A,D,R1),
      pushthrudr (A,D,R2,R1)]
M5 : [pushd (A,D,R1),
      pushthrudr (A,D,R2,R1)]
```

Fig.5.10 獲得されたマクロオペレータ

	P1～P13	P1～P50
PiL2	7	28
STRIPS	16	218
M－STRIPS	104	?

Table5.2 The averages of CPU times (sec)

そして全体的な評価であるCPUタイムなどがすべて減少している．さらに，Table5.2にP1～P13，P1～P50までのCPUタイムの平均値を示す．これからわかるようにM-STRIPSはすでにP1～P13でSTRIPSよりも著しく効率が低下している．PiL2はP50までの平均でSTRIPSの約7.8倍の効率の良さを示しており，少数の有効なマクロオペレータを抽出することによって，かなりの問題の範囲で学習無しシステムよりも効率的なパフォーマンスが実現できることがわかる．

さらに注目すべき点は，Appendix10のP7とP18でPiL2とSTRIPSの解法ステップが同じことである．これはPiL2がそれまで蓄えてきたマクロオペレータがこの二つの問題を解くためには使えず，基本オペレータだけで解いていることを意味する．つまり，この2つの問題はそれまでの問題とは異なったマクロオペレータを必要とする問題であると言える．このような問題に出会った場合，PiL2はマクロオペレータを探索する分だけSTRIPSよりも計算コストが余計にかかり，効率は低下してしまう．Appendix8のP7とP18でPiLの方がCPUタイムが大きいことで，その現象が確認できる．しかし，新しいパターンの問題も一度解決するとマクロオペレータとして学習されるので，Appendix11のP7とP18でPiL2のマクロオペレータが増加しているわけである．

以上のようにPiL2は少数のマクロオペレータだけで学習無し問題解決システムよりも効率の良い問題解決を行うことが可能であることが実験的に証明された．このことはPiL2が完全因果性に基づいて少数の役に立つマクロオペレータのみを選択的に学習可能であり，またその学習方法が問題解決の効率化に有効であることを示している．

5. 6 関連研究との比較

1) STRIPSにおけるMACROPS

学習有りSTRIPS [Fikes 72] において学習されたMACROPSというマクロは三角表という形式で蓄えられる. それに対して, PiL2ではマクロも基本オペレータと同様の構造をしているので, M-STRIPSが学習有りSTRIPSと等価にはならず直接的な比較はできない. しかし, Mintonの指摘 [Minton 85] のとおりMACROPSは重複するもの以外は解決過程の履歴をすべて蓄積していくのでかなり少数の問題 ([Minton 85] によると最高16ステップの問題で20問程度) ですでにマクロの数が増加してしまい, 学習無しシステムよりもパフォーマンスが低下する現象が起こりはじめる. これに対してPiL2は最高ステップ数：28の問題50問でも学習無しSTRIPSよりもほとんどの場合において効率がよいことから, MACROPSを学習するSTRIPSよりも優れていると考えられる.

2) Mintonのマクロオペレータ

Mintonの提案したマクロオペレータには次の二つがある [Minton 85]. 1) S-MACRO：過去の解決履歴を残しておき, 共通する部分シーケンスをマクロとする, 2) T-MACRO：サブゴールに相互作用がある場合に有効なマクロ. 探索木が枝分かれする評価関数の谷の部分が点の近傍でマクロとして生成される.

完全因果性で得られるマクロはS-MACROに対応する. ただし, Mintonの手法には, a) 過去の多くの履歴をすべて残すと膨大な量になる, b) 過去のオペレータシーケンスの共通部分の抽出が容易に出来るのか, などの

問題がある．それに対して，完全因果性の手法は一つのオペレータシーケンスからでも容易にマクロを生成できる点がMintonの手法よりも優れていると思われる．

3) Korfのマクロオペレータ

Korfは「今まで達成したサブゴールを覆すことなくさらに別のサブゴールを達成する」というマクロが存在する問題領域の条件：operator decomposabilityとそのようなマクロを自動生成する手法を提案した[Korf 85]．このマクロの特性は完全因果性によって得られるマクロとは大きく異なっており，比較的狭義なものである．Korfの問題点はproblem solverの問題解決戦略が上記のマクロの特性に強く依存していることと，operator decomposabilityが常に成り立つとは限らないことである．これに対し完全因果性も常に成り立つ保証はないが，prloblem solverの戦略への依存性はない．

また，Korfの方法は基本オペレータと目標状態から考えられるマクロをすべて生成するが，そのうち実際に用いられるマクロはごくわずかである場合は，不要なマクロの探索をしてしまい問題解決の効率が低下する可能性がある．それに対して，PiL2は実際に解決したオペレータシーケンスだけからマクロを生成するので，まったく使われることのないマクロを学習することはない．

4) 操作可能性の基準としての完全因果性

基本オペレータの条件部は問題解決モジュールにとって従来の意味で操作可能なものなので，オペレータシーケンスから得られた説明木は全て

のオペレータの区切れが操作可能である．よってどの区切れで切って一般化してもかまわないが，実際にはその中に役に立たないものが多く含まれている．これはどのように考えればいいのだろうか．その答えはKellerの操作可能性（operationality）についての研究［Keller 87a］が与えてくれる．Kellerはある概念記述が，1）使用可能性（usability）：その概念記述がパフォーマンスシステムにとって使用できる，と2）有効性（utility）：その概念記述がパフォーマンスシステムによって使われたとき，明記された目的についてパフォーマンスが改善される，という2つの条件を満たすとき，操作可能であるとした．このようなKellerの定義からみた場合，全てのマクロの候補は使用可能であるというだけで，その有効性は保証されていない．多くの使用可能なマクロの候補のうち実際にパフォーマンスを改善するもの，つまり有効なものだけが操作可能になるのである．KellerのMETA-LEX［Keller 87b］システムではこの有効性の検証を実際に解法例の集合を解かせて行なっているが，それでは非常に効率が悪い．対照的に完全因果性は実際にテストすることなしに有効なマクロを選択できるヒューリスティックであり，効率的な評価が可能な操作可能性の基準であるといえる．

5. 7 問題点

1）基本オペレータ及び問題状態の記述

完全因果性の成立は，解法例において適用されたオペレータの条件リストと追加リスト中の述語を調べることにより判定される．つまり，完全因果性は基本オペレータがどのように記述されているか，また問題状態がど

のような述語で表現されるのかに依存している．EBGにおいて正しい領域知識が仮定されているのと同様に，完全因果性においては正しい基本オペレータが前提条件になる．もし，基本オペレータや問題状態表現が適切でなかった場合は，有効かつ少数のマクロが得られる保証はない．

2）マクロの増加

PiL2はマクロの選択的学習により，今回与えた50問の問題において5つだけの有効なマクロを学習し，それを用いて学習無しシステムよりも効率のよい問題解決を行えた．しかし，基本オペレータや難しい問題が非常に多く与えられた場合でも，効率の良さを維持できるのだろうか．この問題は今後の課題であるが，希望的な考えでは多くの問題領域に対して完全因果性で得られるマクロの個数は有限であり，いくら多くの基本オペレータと難しい問題を与えようとも学習されるマクロは一定数以上にならず有限であると思われる．実際，3章での1・2次方程式において考えられるすべてのパターンを学習させたときに，得られたマクロが一定数で止まったという結果がでている．しかし，もし上限があるとしてもその上限に達したマクロによる問題解決が基本オペレータだけを用いた問題解決（学習無しシステムの問題解決）よりも，効率が良いのかという疑問が残るが，これについては今のところ検証する術がない．

3）無意味なマクロ

完全因果性はあくまでもヒューリスティックであるので，常に有効なマクロが学習される保証はない．また，前述の正しい基本オペレータや問題状態の記述も常に与えられるとは限らない．今回の実験においては有効か

つ少数のマクロが獲得されたが，もし役に立たないマクロ，さらには4.5で触れたような無意味な探索を誘発する有害なマクロが得られた場合にはシステム自身は対処できない．学習システムを耐久性のあるものにするためにも，このようなマクロが得られた場合にシステムが自動的・半自動的に対処できることが望ましい．

4) 問題を与える順序

今回の実験では問題を易しいものから徐々に難しいものへという順序で与えていった．現在のマクロ抽出方法では，問題を与える順序を換えても同じマクロが得られるとは限らない．特に5.3.2の拘束条件が既存のマクロに依存していることが問題になる．別に同じマクロが得られなくても，どの問題順序でも学習により効率が上がればよいのだが，理論的に問題の順序に独立に同じマクロが学習されることが望ましい．これについても今後の課題であるが5.3.2の拘束条件を同時点で得られるマクロ間にも適用することにより，ある程度実現できると考える．

5. 8 まとめ

完全因果性というヒューリスティックに基づき少数の役に立つマクロオペレータのみを選択的に学習する方法を示し，PiL2を含む3つのシステムを用いた実験によってその有効性を検証した．その結果，完全因果性によるマクロオペレータ学習の手法は方程式解法のみならずロボットの行動計画での学習においても十分有効であることが証明された．また，PiL2の問題状態は述語表現で記述しており汎用性・拡張性に富むと考えられ

る. なお, PiL2はPiL同様SUN3上でK-PROLOG（インタプリタ）を用いてインプリメントされた.

第6章　結論

本論文は機械学習研究において，以下の二つの新しい考えを提示した．

1）マクロオペレータ学習において完全因果性というヒューリスティックを用いることにより，有効なヒューリスティックを選択できること．この完全因果性は新しいヒューリスティックで，人間は因果関係の強いオペレータシーケンスをマクロオペレータとして学習するという仮定に基づいており，各種方程式及びロボットの行動計画においてその有効性が確認された．

2）操作可能な解決可能概念をマクロオペレータを用いて定義し，その概念を用いてEBGがより十分な一般化が可能なことを示した．また，各種方程式の分野においてEBGの過小一般化が改善されることを実験的に確認した．

以上，二点は機械学習研究の基礎的な発展に寄与するものであると考える．

謝 辞

この博士論文は，多くの人々の有形無形の協力及び影響を受けて執筆された．本章では，それらの人々の名前をあげることにより，感謝を表わすことにする．まず，最初に本論文選考の主査であり私の指導教官である辻三郎教授に感謝したい．辻教授は本研究の遂行に必要十分な環境を提供して下さったし，さらに私にとって彼との議論は機知に富んだ刺激的なもので，触発されることが多々あり，本論文に少なからぬ影響を与えたと考える．次に，この決して短いとは言い難い論文を丁寧に読み，適切な指摘をしていただいた副査の豊田順一教授，北橋忠宏教授，安部憲広助教授の各氏に感謝する．彼らの指摘に対する修正がなければ，この論文の質は今よりはるかに低いものとなっていたであろう．

特に，私が5年前AIの研究に進み初めて以来現在に至るまで，常に懇切丁寧な指導を受けた安部憲広助教授に心から感謝したい．彼は私が研究面で最も影響を受けた人物であり，特に私の修士論文の仕事である「電気ドリル分解・組み立てコンサルタントシステム」の研究においては，多大な指導を受けた．もともと，このコンサルタントシステムの基本的なコンセプトは，安部助教授が長年暖めていたもので，私が偶然そのシステムを具体化する仕事に関わったのである．この研究に従事できたことは，私にとって非常に幸運であった．なぜなら，修士の間に曲がりなりにも一つのシステムを設計し，インプリメントすることが出来たからである．この修士における研究が，いろいろな意味で本研究の基盤となっていることは明白である．あらためて，安部助教授に感謝したい．もちろん，博士課程に進んでからも適切な指導を受けたことは言うまでもない．

さらに，修士博士を通じて示唆に富む御指摘をいただいた谷内田正彦助教授，笠井健教授にも感謝したい．また，日頃の議論や渡米の際にお世話になった浅田稔講師，Appendix7を書いてもらった佐古慎二君，また本論文の作成にあたり多大な協力をしてくれた西村和久君をはじめとする辻研FAI（Fundamental AI）グループの皆さん，その他の辻研の皆さんに感謝します．

最後に，大学に9年間も通うという愚行を許し，経済的精神的にも支えてくれた両親と妹に感謝する次第です．

◇ 参考文献 ◇

[Bennett 87] Bennett,S.W. : Approximation in mathematical domain, pp.239-241, Proceeding of IJCAI-87 (1987)

[Boose 84] Boose,J.H. : Personal Construct Theory and the Transfer of Human Expertise, Proceeding of of AAAI-84 (1984)

[Braverman 88] Braverman,M.S.,Russell,S.J. : IMEX : Overcoming Intractability in Explanation Based Learning, pp.575-579, Proceeding of of AAAI-88 (1988)

[Bundy 85] A.バンディ：メタレベル推論と意識, pp.187-203,「AIと哲学」, 産業図書 (1985)

[Buneido 85] 最高水準問題集・高校受験数学, 文英堂 (1985)

[Bunken] アタック2001数学中1・中2, 文研出版

[DeJong 82] Dejong,G. : Automatic Schema Acquisition in a Natural Language Enviroment", p.410-413, Proceeding of AAAI-82 (1982)

[DeJong 86] DeJong,G., Mooney,R. : Explanation-Based Generalization : An Alternative View, pp.145-176, Machine Leaning, Vol.1 No.2 (1986)

[Ellman 88] Ellman,T. : Approximate theory formation : An Explanation-based approach, pp.570-574, Proceeding of of AAAI-88 (1988)

[Ernst 69] Ernst,G. and Newell,A. : GPS : A Case Study

in Generality and Problem Solving, Acadimic Press, New York (1969)

[Fikes 71] R.E.Fikes,N.J.Nilsson : "STRIPS : A New Approach to the Application of Theorem Proving to Problem Solving", pp.189-208, Artificial Intelligence 2 (1971)

[Fikes 72] Fikes,R.E., Hart,P.E., Nilsson,N.J. : Learning and Executing Generelized Robot Plans, pp.251-288, Artificial Intelligence 3 (1972)

[Furukawa 86] 古川, 武脇 : 数式処理における学習」, p.155-169, 知識の学習メカニズム, 共立出版 (1986)

[Haraguchi 86] 原口, 有川 : 類推の定式化とその実現, pp.132-139, 人工知能学会誌, Vol.1 No.1 (1986)

[Hirsh 87] Hirsh,H. : Explanation-Based Generalization in a Logic-Programing Environment, pp.221-227, Proceeding of IJCAI-87 (1987)

[Hopfield 85] Hopfield,J.J. and Tnak,D.W. : Neural Computation of Decision in Optimization Problems, pp.147-152, Biological Cybernetics, Vol.52 (1985)

[Hori 86] 堀 : 学習研究の期待・課題・将来 -学習システムの種と花-, 計測と制御, Vol.25, No.9 (1986)

[Iba 85] Iba,G.A. : Learning by discovering macros in puzzlesolving, pp.640-642, Proceeding of IJCAI-85 (1985)

[Kahn 85] Kahn,G.,Nowlan,S and McDermott,J. : Strategies for Knowledge Acquisition, pp.511-522, IEEE Trans. PAMI,

Vol.PAMI-7, No.5 (1985)

[Kasuga 75] 春日正文編：公式集（モノグラフ24），科学新興社（1975）

[Kedar-Cabelli 87] Kedar-Cabelli,S.T. : Formulating Concept According to Purpose, pp.477-481, Proceeding of AAAI-87 (1987)

[Keller 87a] Keller,R.M. : Defining Operationality for Explanation-Based Learning, pp.482-487, Proceeding of AAAI -87 (1987)

[Keller 87b] Keller,R.M. : Concept Learning in Context, pp.91-102, Proceeding of International Workshop on Machine Learning (1987)

[Kibler 83] Kibler,D., Porter,B. : Perturbation : A means for guiding generalization, pp.415-418, Proceeding of IJCAI -83 (1983)

[Korf 83] Korf,R.E. : Operator Decomposability : A new type of problem structure, pp.206-209, Proceeding of AAAI-83 (1983)

[Korf 85] Korf,R.E. : A Weak Method for Learning, pp.35 -77, Artificail Intelligence, Vol.26 No.1 (1985)

[Korf 87] Korf,R.E. : Planning as Search : A Quantitative Approach, pp.65-88, Artificial Intelligence, Vol.33 No.1 (1987)

[Laird 84] Laird,J.E.,Rosenbloom,P.S. and Newell,A. : Towards Chunking as a General Learning Mechanism, pp.188-192, Proceeding of of AAAI-84 (1984)

[Laird 86] Laird,J.E.,Rosenbloom,P.S. and Newell,A. : Chunking in Soar : The Anatomy of a General Learning Mechanism, pp.11-46, Machine Learning, Vol.1, No.1 (1986)

[Lebowitz 85] Lebowitz,M. : Concept learning in a rich input domain : Generalization-based menory. pp.193-214, in R. S.Michalski,J.G.Carbonell & T.M.Mitchell (Eds.),Machine learning : An artificialintelligence approach,Vol.2. Los Altos,CA : Morgan Kaufmann (1985)

[Lenat 83] Lenat,D.B. : The Role of Heuristics in Learning by Discovery : Three Case Studies, pp.163-190, in Machine Learning-An artificial intelligence approach vol.1, Tiago Pub. Co., Palo Alto (1983)

[Mahadevan 85] S.Mahadevan : Verification-Based Learning : A Generalization Strategy for Infering Problem-Reduction Methods, pp.616-623, Proceeding of IJCAI-85 (1985)

[McCulloch 43] McCulloch,W.S. and Pitts,W. : A logical calculus of the ideas immanent in nervous activity, pp.115 -133, Bullet. Math. Biophysics 5 (1943)

[Michalski 83] Michalski,R.S. : A Theory and Methodology of Inductive Learning, pp.111-161, Artificial Intelligence 20 (1983)

[Miller 56] Miller,G.A. : The Magic Number Seven Plus or Minus Two : Some Limits on Our Capacity for Processing Information, pp.81-97, Psychological Review, Vol.63 (1956)

[Minton 84] Minton,S. : Constraint-Based Generalization, pp.251-254, Proceeding of AAAI-84 (1984)

[Minton 85] Minton,S. : Selectively Generalizing Plans for Problem-Solving, pp.596-599, Proceeding of of IJCAI-85 (1985)

[Minton 87] Minton,S. and Carbonell,J.G. : Strategies for Learning Control Rules : An explanation-based Approach, pp.228-235, Proceeding of IJCAI-87 (1987)

[Mitchell 78] Mitchell,T.M. : Version Spaces : An approach to concept learning, Ph.D. dissertation, Stanford University (1978)

[Mitchell 81] Mitchell,T.M. Utgoff,P.E. Bernard Nudel, Ranan Banerji : Learning problem-solving heuristics through practice, pp.127-134, Proceeding of of IJCAI-81 (1981)

[Mitchell 83a] Mitchell,T.M., Utgoff,P.E. & Banerji,R. : Acquring and Refining Problem-Solving Heurristic, pp.163-190, in Machine Learning-An artificial intelligence approach vol.1, Tiago Pub. Co., Palo Alto (1983)

[Mitchell 83b] Mitchell,T.M. : Learning and Problem Solving, pp.1139-1151, Proceeding of of IJCAI-83 (1983)

[Mitchell 85] Mitchell,T.M., Mahadevan,S. and Steinberg,L. I. : LEAP : A Learning Apprentice for VLSI, pp.573-580, Proceeding of of IJCAI-85 (1985)

[Mitchell 86] Mitchell,T.M., Keller & Kadar-Cabelli : Explanation -Based Generalization : A Unifying View, pp.47-80, Machine

Learnig, Vol.1 No.1 (1986)

[Mooney 86] R.J.Mooney, S.W.Bennett : A Domain Independent Explanation-based Generalizer, Proceeding of AAAI-86 (1986)

[Mostow 87] Mostow,J., Bhatnagar,N. : Failsafe : A Floor Planner that Uses EBG to Learn from its Failure, pp.249-255, Proceeding of IJCAI-87 (1987)

[Numao 86] 沼尾, 志村：プロダクションシステムNatにおける学習機能, pp.1669-1670, 情報処理学会第33回全国大会 (1986)

[Numao 88] 沼尾正行：説明に基づく一般化 -領域固有の知識を用いたアプローチ-, pp.704-711, 人工知能学会誌, Vol.3, No.6 (1988)

[Polanyi 66] Polanyi,M. : The Tacit Dimention, Routledge & Kegan Paul Ltd.,(1966) 邦訳「暗黙知の次元」(佐藤敬三訳, 紀ノ国屋書店)

[Porter 85] Porter,B.W, and Kibler,D.F. : A Comparison of Analytic and Experimental Goal Regression for Machine leaning, pp.555-559, Proceeding of IJCAI-85 (1985)

[Ritchie 84] Ritchie,G.D and Hanna,F.K. : AM : A Case Study in AI Methodology, pp.249-268, Artificial Intelligence, Vol.23 (1984)

[Rosenbloom 86] Rosenbloom,P.S. and Laird,E.J. : Mapping Explaclnation-Based Generalization onto Soar, pp.561-567, Proceeding of AAAI-86 (1986)

[Rumelhart 86] Rumelhart,D.E., Hinton,G.E. and Williams, R.J. : Learning Internal Representations by Error Propagation,

pp.318-362, Parallel Distributed Processing, Vol.1, The MIT Press Cambridge, London, England (1986)

[Salvik 87] Salvik,J.W., Dejong,G.F. : An Explanation-based Approach to Generalizing Number, pp.236-238, Proceeding of IJCAI-87 (1987)

[Sato 84] 佐藤忠：モノグラフ対数関数，科学新興社 (1984)

[Serge 87] Segre,A.M. : On the Operationality／Generality Trade-Off in Explanation-Based Learning, pp.242-248, Proceeding of IJCAI-87 (1987)

[Silver 83] Silver,B. : Learning Equation Solving methods from Examples, pp.429-431, Proceeding of IJCAI-83 (1983)

[Silver 86a] Silver,B. : Meta-Level Inference, North Holland (1986)

[Silver 86b] Silver,B. : Precondition Analysis : Learning Control Information, In R.S.Michalski,J.G.Carbonell & T.M. Mitchell (Eds.),Machine learning : An artificialintelligence approach,Vol.2. Los Altos,CA : Morgan Kaufmann (1986)

[Utgoff 86] Utgoff, P.E. : Sift of Bias for Inductive Concept Learning, pp.107-148, In R.S.Michalski,J.G.Carbonell & T.M. Mitchell (Eds.),Machine learning : An artificial intelligence approach,Vol.2. Los Altos,CA : Morgan Kaufmann (1986)

[Watanabe 86] 渡辺，岩本，山之内，松田：VILLA：VLSI設計知識獲得システム，電子情報通信学会研究会，AI-86-1 (1986)

[Watanabe 88] 渡辺正信：知識獲得と学習，第二回人工知能学会全国

大会チュートリアル講演資料 (1988)

[Winston 75] Winston,P.H. : Learning Structural Description from Examples, PhDthesis, in The Psychology of Computer Vision, edited by P.H.Winston, McGraw-Hill Book Company, New York (1975)

[Winston 80] Winston,P.H. : Learning and reasoning by Analogy, Commun ACM, Vol.23 (1980)

[Winston 83] Winston,P.H. : Learning New Principles from Precedents and Exercises, pp.321-350, Artificial Intelligence, Vol.19 (1983)

[Yamada 87a] 山田, 安部, 辻 : 条件連鎖に基づく一般化による学習, pp.81-84, 第1回人工知能学会大会資料 (1987)

[Yamada 87b] 山田, 安部, 辻 : 解法例からの問題解決知識の獲得 -1次方程式・不等式の場合-, 情報処理学会, 人工知能と知識工学研究会資料, 87-AI-54 (1987)

[Yamada 87c] 山田, 安部, 辻 : 問題解決における学習システム : PiL, 情報処理学会,「人工知能システムの枠組み」シンポジウム論文集 (1987)

[Yamada 88a] 山田, 安部, 辻 : 問題解決における戦略知識学習システム : PiL -1次方程式・不等式でのケーススタディ-, 人工知能学会誌, pp.206-215, Vol.3 No.2 (1988)

[Yamada 88c] 山田, 安部, 辻 : 問題解決における戦略学習システム : PiL -直接解決可能性に基づく一般化-, 情報処理学会, 人工知能と知識工学研究会資料, 88-AI-56 (1988)

[Yamada 88d] 山田, 辻, 安部 : 直接解決可能性に基づく一般化 : DSBG

-操作可能なSOLVABLE概念の定義付け-，pp.783-791，人工知能学会誌，Vol.3 No.6 (1988)

[Yamada 88e] 山田，辻：完全因果性によるマクロオペレータの選択的学習，情報処理学会，人工知能と知識工学研究会資料，88-AI-60 (1988)

[Yamada 89] 山田，辻：完全因果性によるマクロオペレータの選択的学習，人工知能学会誌，Vol.4 No.3 (1989) (to appear)

◇ 研究業績 ◇

[論文]

山田, 安部, 辻：電気ドリル分解・組立てコンサルタント・システム, 人工知能学会誌, Vol.1 No.1 (1986)

山田, 安部, 辻：問題解決における戦略知識学習システム：PiL -1次方程式・不等式でのケーススタディ-, 人工知能学会誌, Vol.3 No.2 (1988)

山田, 辻, 安部：直接解決可能性に基づく一般化：DSBG -操作可能なSOLVABLE概念の定義づけ-, 人工知能学会誌, Vol.3 No.6 (1988)

山田, 辻：完全因果性によるマクロオペレータの選択的学習, 人工知能学会誌, Vol.4 No.3 (1989) to appear

[口頭発表]

Abe, N., Yamada, S. and Tsuji, S. : Consulting system which detects and undoes erroneous operations by novices, Proceeding of Third International Symposium on Optical and Optoelectronic Applied Sciences and Engineering (1986)

Yamada, S., Abe, N. and Tsuji, S. : Construction of a Consulting

System from Structural Description of a Mechanical Object, pp.1413-1418, Proceeding of 1987 IEEE Interenational Conference on Robotics & Automation (1987)

Yamada, S., Abe, N. and Tsuji, S. : Constructing a Consulting System from Structural Description of a Mechanical Object, Proceeding of AVIGNON'89 ; Ninth International Workshop on Expert Systems & their Application (1987)

山田, 安部, 辻：電気ドリル分解コンサルタントシステムのための衝突検出, p.224, 昭和60年度電子通信学会総合全国大会資料集 (1985)

山田, 安部, 辻：電気ドリル分解・組立てコンサルタント・システムの作成, pp.77-86, 情報処理学会「知識情報処理シンポジウム」資料集 (1985)

山田, 安部, 辻：機械部品の分解・組立てコンサルタント・システムのインタフェースについて, pp.65-72, 計測自動制御学会　第1回ヒューマン・インタフェース・シンポジウム資料集 (1985)

山田, 安部, 辻：電気ドリル分解・組立てコンサルタント・システム, 電気関連学会関西支部連合大会資料集 (1985)

山田, 安部, 辻：電気ドリル分解・組立てコンサルタント・システムにおけるマン・マシン・インタフェース, p.216, 昭和61年度電子通信学会総

合全国大会資料集（1986）

山田，安部，辻：機能モデルによる機械の故障診断計画，pp.1411-1412，情報処理学会第34回（昭和62年前期）全国大会（1987）

山田，安部，辻：条件連鎖に基づく一般化による学習，pp.81-84，昭和62年度人工知能学会全国大会（第1回）（1987）

山田，安部，辻：解法例からの問題解決知識の獲得　-1次方程式・不等式の場合-，情報処理学会「知識工学と人工知能」研究会報告No.54　87-AI-54-7（1987）

山下，山田，安部，辻：機械部品分解・組立コンサルテーションにおけるマン・マシンインタフェース，pp.1797-1798，情報処理学会第35回（昭和62年度後期）全国大会資料集（1987）

山田，安部，辻：機械部品の3Dモデル空の運動伝達抽出，pp.1527-1528，情報処理学会第35回（昭和62年度後期）全国大会資料集（1987）

山田，安部，辻：問題解決における学習システム：PiL，pp.131-140，情報処理学会「人工知能システムの枠組み」シンポジウム資料集（1987）

山田，安部，辻：問題解決における戦略学習システム：PiL　-直接解決可能性に基づく一般化-，情報処理学会「知識工学と人工知能」研究会報告

No.56 88-AI-56-14 (1988)

山田, 安部, 辻：問題解決における戦略学習システム：PiL -各種方程式への拡張-, pp.1663-1664, 情報処理学会第36回（昭和63年前期）全国大会（1988）

山下, 山田, 安部, 辻：機械部品分解コンサルテーションシステムにおける不測事態の対処 -ユーザモデル・対象の履歴とその利用-, pp.1529-1530, 情報処理学会第36回（昭和63年前期）全国大会（1988）

山田, 辻：完全因果性によるマクロオペレータの選択的学習, 情報処理学会「知識工学と人工知能」研究会報告No.60 88-AI-60-9 (1988)

< Appendix1 > 基本オペレータ

rule000 : LHS = RHS → RHS = LHS, [], []
rule010 : LHS = RHS → LHS + A = RHS + A, [], [[not_zero, A]]
rule020 : LHS = RHS → LHS * A = RHS * A, [], [[not_one, A]]
rule030 : LHS = RHS → LHS/A = RHS/A, [], [[not_zero, A]]
rule040 : LHS > RHS → LHS + A > RHS + A, [], [[not_zero, A]]
rule041 : LHS > RHS → LHS * A > RHS * A, [], [[plusp, A]]
rule042 : LHS > RHS → LHS/A > RHS/A, [], [[plusp, A]]
rule043 : LHS > RHS → LHS * A < RHS * A, [], [[minusp, A]]
rule044 : LHS > RHS → LHS/A < RHS/A, [], [[minusp, A]]
rule045 : LHS < RHS → LHS + A < RHS + A, [], [[not_zero, A]]
rule046 : LHS < RHS → LHS * A < RHS * A, [], [[plusp, A]]
rule047 : LHS < RHS → LHS/A < RHS/A, [], [[plusp, A]]
rule048 : LHS < RHS → LHS * A > RHS * A, [], [[minusp, A]]
rule049 : LHS < RHS → LHS/A > RHS/A, [], [[minusp, A]]
rule050 : LHS >= RHS → LHS + A >= RHS + A, [], [[not_zero, A]]
rule051 : LHS >= RHS → LHS * A >= RHS * A, [], [[plusp, A]]
rule052 : LHS >= RHS → LHS/A >= RHS/A, [], [[plusp, A]]
rule053 : LHS >= RHS → LHS * A =< RHS * A, [], [[minusp, A]]
rule054 : LHS >= RHS → LHS/A =< RHS/A, [], [[minusp, A]]
rule055 : LHS =< RHS → LHS + A =< RHS + A, [], [[not_zero, A]]
rule056 : LHS =< RHS → LHS * A =< RHS * A, [], [[plusp, A]]
rule057 : LHS =< RHS → LHS/A =< RHS/A, [], [[plusp, A]]
rule058 : LHS =< RHS → LHS * A >= RHS * A, [], [[minusp, A]]
rule059 : LHS =< RHS → LHS/A >= RHS/A, [], [[minusp, A]]
rule060 : LHS < RHS → RHS > LHS, [], []
rule061 : LHS > RHS → RHS < LHS, [], []
rule062 : LHS =< RHS → RHS >= LHS, [], []
rule064 : LHS >= RHS → RHS =< LHS, [], []
rule100 : log (A, A) → 1, [], []
rule101 : log (A, 1) → 0, [], []
rule102 : log (A, B * C) → log (A, B) + log (A, C), [], []
rule103 : log (A, B) + log (A, C) → log (A, B * C), [], []
rule104 : log (A, B/C) → log (A, B) − log (A, C), [], []
rule105 : log (A, B) − log (A, C) → log (A, B/C), [], []
rule106 : log (A, exp (B, C)) → C * log (A, B), [], []
rule107 : C * log (A, B) → log (A, exp (B, C)), [], []
rule108 : log (A, B) → log (C, B) /log (C, A), [], []
rule109 : log (C, B) /log (C, A) → log (A, B), [], []
rule110 : log (C, A) = log (C, B) → A = B, [], []

```
rule111 : log (A, B) =C → B = exp (A, C), [], []
rule112 : log (R1, R2) → R3,
              [[real_number, R1], [real_number, R2]],
              [[equal, R3, [log, R1, R2]], [real_number, R3]]
rule113 : log (A, A) − log (A, A) → 0, [], []
rule120 : exp (A, 0) → 1, [], []
rule121 : exp (A, B) * exp (A, C) → exp (A, B + C), [], []
rule122 : exp (A, B) /exp (A, C) → exp (A, B − C), [], []
rule123 : exp (exp (A, B), C) → exp (A, B * C), [], []
rule124 : exp (A * B, C) → exp (A, C) * exp (B, C), [], []
rule125 : exp (A/B, C) → exp (A, C) /exp (B, C), [], []
rule126 : exp (A, B) = exp (A, C) → B = C, [], []
rule127 : exp (R1, R2) → R3,
              [[real_number, R1], [real_number, R2]],
              [[equal, R3, [* *, R1, R2]], [real_number, R3]]
rule128 : exp (R1, A) = R2 → A = log (R1, R2),
              [[real_number, R1], [real_number, R2]],
              []
rule200 : (B + C) * A → B * A + C * A, [], []
rule210 : A * B + A * C → (B + C) * A, [], []
rule211 : B * A * A + C * A * A → (B + C) * A * A, [], []
rule220 : 0 + A → A, [[not_real_number, A]], []
rule230 : 0 * A → 0, [], []
rule240 : 0/A → 0, [], []
rule250 : 1 * A → A, [], []
rule260 : A/1 → A, [], []
rule270 : R1 + R2 → 0,
              [[real_number, R1], [real_number, R2],
                [equal, R2, [*, R1, − 1]], [equal, R1, [*, R2, − 1]]], []
rule280 : A/A → 1, [[not_zero, A]], []
rule300 : R1 + R2 → R3,
              [[real_number, R1], [real_number, R2]],
            [[equal, R3, [+, R1, R2]], [real_number, R3]]
rule310 : R1 * R2 → R3,
              [[real_number, R1], [real_number, R2]],
            [[equal, R3, [*, R1, R2]], [real_number, R3]]
rule320 : R1/R2 → R3,
              [[real_number, R1], [real_number, R2], [not_zero, R2]],
              [[equal, R3, [/, R1, R2]], [real_number, R3]]
rule330 : A/B + C/D → (A * D + B * C) / (B * D), [], []
rule340 : (A * B) /C → (A/C) * B, [], []
rule341 : (A * B) /C → (B/C) * A, [], []
```

```
rule342 : (A * B * C) /D → (A/D) * B * C, [], []
rule343 : (A * B * C) /D → (B/D) * A * C, [], []
rule345 : A * (B/C) → (A/C) * B, [], []
rule346 : A * (B/C) → (A * B) /C, [], []
rule350 : (A + B) /C → (A/C) + (B/C), [], []
rule351 : (A + B + C) /D → (A/D) + (B/D) + (C/D), [], []
rule352 : (A + B + C) * D → (A * D) + (B * D) + (C * D), [], []
rule400 : A * A + 2 * B * A + B * B → (1 * A + B) * (1 * A + B), [], []
rule410 : A * A - 2 * B * A + B * B → (1 * A - B) * (1 * A - B), [], []
rule420 : A * A - B * B → (1 * A - B) * (1 * A - B), [], []
rule430 : A * B → C * (A/C) * B, [], []
rule440 : A * A = B → A = SR, [], [[equal, SR, [plus_minus_sr, B]]]

rule1000 : 1 * A = R → finish,
              [[alphabet, A], [real_number, R]], []
rule1005 : A = R → finish,
              [[alphabet, A], [real_number, R]], []
rule1010 : 1 * A > R → finish,
              [[alphabet, A], [real_number, R]], []
rule1020 : 1 * A < R → finish,
              [[alphabet, A], [real_number, R]], []
rule1030 : 1 * A >= R → finish,
              [[alphabet, A], [real_number, R]], []
rule1040 : 1 * A =< R → finish,
              [[alphabet, A], [real_number, R]], []
```

< Appendix2 > 訓練例

1次方程式 ：85問

```
# 5 * x = (-5)
  ((-1) /3) * y = 3
  0.2 * x = (-3)
  (-12) * x ='4/9'
# 10 = 2 * x
# 2 * x + 6 = 12
  (-5) + 1 * x = (-2)
  1 * y + 0.2 = (-0.4)
  '1/2' + 1 * x ='1/3'
  7 * x + 4 = 18
  3 * x + 5 = (-4)
  12 - 3 * x = (-15)
# 15 = 5 * x - 5
  (-7) = (-3) - 4 * x
  2 * x - 0.7 = (-0.3)
  1.8 - 3 * x = (-3.6)
  0.5 = 1.5 - 1 * x
  1 * x -'1/5' = (-1)
  '2/5' - 11 * x ='1/3'
  '1/4' = 14 * x +'1/3'
# 6 * x + 3 = 4 * x - 5
  7 * x - 5 = 4 * x + 13
  1 * x - 15 = (-7) - 3 * x
  5 * x - 4 - 7 * x = 0
  12 + 4 * x = 12 * x - 84
  9 * x + 52 = 6 * x + 67
  3 * y - 250 = 12 * y - 160
  8 * y - 5 - 2 * y = 4 * y + 3
# 3 * (4 * x - 2) = 5 * x + 8
  4 * x - 3 = 2 * (1 * x + 3) + 1
  2 * (2 * y - 3) + 5 = 5 * y - 6
  4 * (3 * x - 2) = 15 - (1 * x - 3)
  (-2) * (1 * x + 3) = 4 - (1 * x - 5)
  (-3) * (4 - 1 * x) - 4 * (2 * x - 3) = 0
  1.4 * x - 3.3 = 0.9 * x - 2.8
  0.16 + 0.32 * x = 0.24 * x - 0.48
  0.95 - 0.09 * x = 0.3 * x - 1
```

0.3 * (1 * x - 2) - 0.4 = (- 1.2) * x
1.2 * (3 - 5 * x) - 0.3 * (2 * x + 1) = 0
2 + '1/2' * x = '1/6' * x
'1/4' * x + 2 = '1/2' * x - 4
2 * x + (12 * x - 6) /3 = 9
1 * x - (1 * x - 1) /3 = (- 3)
1 * x - (2 * x - 1) /5 = (- 4)
2 * x - (1 * x - 1) /3 = 7
1 * x - (3 * x - 4) /4 = 2 * x - 13
(1 * x) /3 - (1 * x - 1) /5 = 1
'1/2' * (1 * x - 3) - 2 * x = 6
(2 * x + 5) /3 = (1 * x - 5) /4
4 * x - 7 = 1 * x - 1
2 * x - 3 = 5 * x + 9
2 * x - 12 = 8 - 3 * x
15 * x - 28 - 8 * x = 0
28 - 18 * x = 14 * x - 68
(- 8) * x - 41 = 5 * x + 63
5 * y - 162 = 45 - 18 * y
(- 6) * y - 4 = 5 * y - 12 - 9 * y
2 * x - 3 * (1 * x - 1) = (- 2)
3 + 2 * (1 * x - 2) = 1 * x + 2
5 * (2 * x - 33) - 3 * (1 * x - 3) = 4 - 1 * x
0.4 * x + 0.7 = 0.9 * x - 1.3
1 + 1.06 * x = 0.82 * x - 0.2
0.16 * (2 * y - 1) = 1 - 0.3 * (5 - 1 * y)
2 * y - '1/3' * (5 * y - 2) = 1
'1/3' * (1 * x + 4) = 1/2
2 - (1 * x - 4) /5 = 3 * x
(1 * x - 4) /3 = 1 - (1 * x + 2) /6
(2 * x - 3) /4 - (3 * x + 4) /3 = ' - 7/12'
2 * x - 11 = (3 * x - 1) /2 - 1 * x
0.2 * (0.3 * x - 0.4) = 0.1
3.8 * x - 1.2 * (1 * x - 0.5) = 0.8 * (2 * x - 0.5)
0.3 * (1 * x - 0.9) - 0.57 = 1.3 * (2 * x - 1)
'1/4' * x - (2 * x - 1) /3 = '5/6' * (1/2 - '4/5' * x) - 1
2 * (2 * x - 3) /3 + (1 * x - 8) /5 + '8/15' = 0
2 * x - (4 - 3 * x) /9 = (1 * x - 2) /3
(1 * x - 3) /4 - (3 * x - 5) /8 = 1 + (1 * x + 4) /2
(2 - 1 * x) /6 + (4 * x - 1) /3 = (5 * x - 4) /9 - 1/2
(1 * x - 1) /2 - 3 * (1 * x - 2) /5 = 1 - (1 * x - 5) /2
0.5 * x - 0.8 * (1 * x - 1.5) = 0.4 * (3 * x - 7)

1 − (1 ＊ x − 3) ／2 = 1 ＊ x − (1 ＊ x − 2) ／3
(1 ＊ x + 1) ／3 − (2 ＊ x − 3) ／5 = 7／10
(2 ＊ x − 2) ／4 = (1 ＊ x + 3) ／3 − 1 ＊ x − 5
(1 ＊ x + 3) ／2 − (1 ＊ x − 1) ／6 = 1 − (1 ＊ x + 4) ／3
5 ＊ (1 ＊ x − 1) ／6 − 3 ＊ (1 ＊ x + 1) ／2 = (3 ＊ x − 5) ／4
(4 ＊ x + 10) ／2 − 3 ＊ (1 ＊ x + 3) ／5 = 3 ＊ x − 5 ＊ (2 − 1 ＊ x) ／4

2次方程式 ：211問

x ＊ x = 25
x ＊ x = 49
x ＊ x = '4／9'
x ＊ x = 2
x ＊ x = 8
x ＊ x = 12
x ＊ x − 9 = 0
x ＊ x − 36 = 0
x ＊ x − 6 = 0
x ＊ x − '5／16' = 0
x ＊ x − 16 = 0
x ＊ x − '3／4' = 1
2 ＊ x ＊ x = 8
2 ＊ x ＊ x = 18
3 ＊ x ＊ x − 9 = 0
2 ＊ x ＊ x − 32 = 0
3 ＊ x ＊ x − 21 = 0
2 ＊ x ＊ x + 5 = 41
2 ＊ x ＊ x − 6 = 42
5 ＊ x ＊ x − 9 = 3
8 ＊ x ＊ x − 50 = 10
'1／3' ＊ x ＊ x − 6 = 2
'3／2' ＊ x ＊ x + 2 = 5
'5／6' ＊ x ＊ x − '3／4' = '1／3'
(1 ＊ x + 3) ＊ (1 ＊ x + 3) = 16
(1 ＊ x − 2) ＊ (1 ＊ x − 2) = 4
(1 ＊ x + 4) ＊ (1 ＊ x + 4) = 9
(1 ＊ x − 5) ＊ (1 ＊ x − 5) = 100
(1 ＊ x − 1) ＊ (1 ＊ x − 1) − 9 = 0
(1 ＊ x − 6) ＊ (1 ＊ x − 6) − 49 = 0
x ＊ x + 6 ＊ x = 5
x ＊ x + 2 ＊ x = 1
x ＊ x − 4 ＊ x + 2 = 0

x * x − 6 * x − 8 = 0
x * x + 1 * x − 5 = 0
x * x − 5 * x − 2 = 0
x * x + 3 * x + 1 = 0
x * x + 5 * x − 2 = 0
x * x − 3 * x + 2 = 0
x * x − 2 * x − 1 = 0
x * x + 2 * x − 5 = 0
y * y − 6 * y + 6 = 0
2 * x * x + 10 * x + 6 = 0
2 * x * x − 3 * x + 1 = 0
3 * x * x + 2 * x − 1 = 0
3 * x * x − 2 * x − 1 = 0
2 * x * x + 3 * x − 2 = 0
3 * x * x − 3 * x − 7 = 0
(1 * x + 2) * (1 * x + 3) = 0
(1 * x + 4) * (1 * x − 5) = 0
x * (1 * x − 2) = 0
(1 * x − 7) * (1 * x − 7) = 0
(4 * x − 2) * (4 * x − 3) = 0
(1 * x − 4) * (1 * x + 7) = 0
x * x − 7 * x = 0
x * x + 3 * x = 0
2 * x * x + 10 * x = 0
5 * x * x − 3 * x = 0
3 * x * x = (−2) * x
x * x = 6 * x
x * x + 4 * x + 4 = 0
x * x + 8 * x + 16 = 0
x * x + 20 * x + 100 = 0
x * x + 36 = (−12) * x
y * y + 2 * y = (−1)
x * x = (−6) * x − 9
x * x − 2 * x + 1 = 0
x * x − 6 * x + 9 = 0
x * x − 4 * x + 4 = 0
x * x + 16 = 8 * x
y * y − 10 * y = (−25)
(−20) * y + 100 = 2 * y * y
x * x − 4 * x − 5 = 0
x * x − 5 * x + 6 = 0
x * x + 1 * x − 12 = 0

x＊x＋6＊x－16＝0
x＊x－1＊x－56＝0
x＊x＋3＊x＋2＝0
x＊x－1＊x＝0
2＊x＊x＋16＊x＝0
2＊x＊x－1＊x＝0
x＊x＝（－7）＊x
5＊x＊x＝3＊x
p＊p＋2＊p＋1＝0
y＊y＋6＊y＋9＝0
m＊m＋8＊m＋16＝0
t＊t＋12＊t＝（－36）
p＊p－4＊p＋4＝0
y＊y－20＊y＋100＝0
25－10＊x＝（－1）＊x＊x
49－14＊x＋x＊x＝0
x＊x＋12＊x＝（－20）
x＊x－2＊x－15＝0
x＊x＋1＊x－6＝0
x＊x－3＊x－10＝0
x＊x＝5＊x＋6
x＊x＝（－8）＊x－12
x＊x＋5＊x＝14
x＊x－5＊x＋2＝0
x＊x－2＊x－24＝0
y＊y－2＊y－5＝0
x＊x－27＝0
x＊x＋3＊x＋1＝0
x＊x－15＊x＋54＝0
x＊x＋1＊x－30＝0
y＊y＋4＊y－5＝0
24－x＊x＝0
x＊x＋6＊x＋2＝0
t＊t－4＊t－12＝0
m＊m－4＊m＋4＝0
3＊x＊x－75＝0
4＊x＊x－60＝0
25＊x＊x－49＝0
16＊x＊x－3＝0
8＊x＊x－5＝11
3＊x＊x－2＝3
x＊x－8＊x－1＝0

```
x*x+6*x-16=0
x*x-3*x-1=0
x*x-5*x+1=0
2*x*x-5*x-4=0
3*x*x+2*x-7=0
x*x+11*x-25=0
x*x-15*x+15=0
x*x-9*x-4=0
x*x+3*x-108=0
y*y-18*y+25=0
p*p+13*p-12=0
3*x*x+7*x+1=0
2*x*x+7*x-2=0
4*x*x-9*x+2=0
3*x*x-2*x-4=0
5*x*x+6*x-3=0
6*x*x-13*x-6=0
(1*x-2)*(1*x-2)=5
(1*x+1)*(1*x+1)-3=0
(1*x+3)*(1*x+3)=27
(1*x+5)*(1*x+5)=8
'1/2'*x*x+3*x-2=0
'1/6'*x*x-'1/2'*x-'1/3'=0
'1/12'*x*x+'1/3'*x-'1/4'=0
0.1*x*x-0.8*x-0.2=0
0.1*x*x+0.4*x-1.2=0
2*x*(4*x-1)=10
# 5*(3*x*x-2*x+9)=3
5*(1*x-2)*(1*x-2)=125
7*(1*x+10)*(1*x+10)-63=0
# 10*x*x-3*x-9=4*x*x-36*x+6
# 10*x*x-3*x=4*x*x-36*x+6
# 10*x*x-9=4*x*x-36*x+6
# (-3)*x-9=4*x*x-36*x+6
# 10*x*x=4*x*x-36*x+6
# (-9)=4*x*x-36*x+6
# (-3)*x=4*x*x-36*x+6
# 10*x*x-3*x-9=4*x*x-36*x
# 10*x*x-3*x=4*x*x-36*x
# 10*x*x-9=4*x*x-36*x
# (-3)*x-9=4*x*x-36*x
# 10*x*x=4*x*x-36*x
```

```
#  (-9) =4*x*x-36*x
#  (-3) *x=4*x*x-36*x
   10*x*x-3*x-9=4*x*x+6
#  10*x*x-3*x=4*x*x+6
#  10*x*x-9=4*x*x+6
#  (-3) *x-9=4*x*x+6
#  10*x*x=4*x*x+6
#  (-9) =4*x*x+6
#  (-3) *x=4*x*x+6
   10*x*x-3*x-9= (-36) *x+6
   10*x*x-3*x= (-36) *x+6
#  10*x*x-9= (-36) *x+6
    (-3) *x-9= (-36) *x+6
#  10*x*x= (-36) *x+6
    (-9) = (-36) *x+6
    (-3) *x= (-36) *x+6
   10*x*x-3*x-9=4*x*x
   10*x*x-3*x=4*x*x
#  10*x*x-9=4*x*x
    (-3) *x-9=4*x*x
#  10*x*x=4*x*x
#  (-9) =4*x*x
#  (-3) *x=4*x*x
   10*x*x-3*x-9=6
   10*x*x-3*x=6
   10*x*x-9=6
    (-3) *x-9=6
   10*x*x=6
   10*x*x-3*x-9= (-36) *x
   10*x*x-3*x= (-36) *x
   10*x*x-9= (-36) *x
    (-3) *x-9= (-36) *x
   10*x*x= (-36) *x
    (-9) = (-36) *x
#  (-3) *x= (-36) *x
#  ((1*x+3) * (4*x+2)) /2=1*x*x
   2* (1*x-1) * (1*x-1) -30=0
    (6-1*x) * (6-1*x) -50=0
   3* (1-1*x) * (1-1*x) -54=0
   6* (1*x+7) * (1*x+7) =7
   5* (1*x-4) * (1*x-4) -12=0
   3* (1*x-1) * (1*x-1) -27=0
```

2 * x * x + 5 * x = 10 * x + 20
x * (2 * x − 6) = 5 * (1 * x − 6)
5 * x * x − 3 = 3 * x * x + 4 * x
1 * x * x − 3 = x * (2 * x − 7)
2 * x * x − 10 * x + 12 = 10 * x − 20
(1 * x − 2) * (1 * x − 3) = 5 * (1 * x − 2)
(1 * x − 3) * (1 * x − 4) = 2 * (8 − 1 * x)
4 * (x * x − 2) = 3 * x * x − 2 * x
2 * (2 * x − 1) * (3 * x − 1) = 6
2 * (1 * x − 1) * (1 * x − 1) = (1 * x − 1) * (1 * x + 3)

分数方程式 ：35問

3／ (4 * x − 5) = 10
(7 * x) ／ (12 * x + 1) = 3
(3 * x + 20) ／ (5 * x − 2) = 10
(2 * x * x + 5) ／ (10 * x + 2) = 20
(10 * x * x − 3 * x − 9) ／ (12 * x + 2) = 3
(4 * x * x + 3 * x) ／ (10 * x − 6) = 4
(3 * x * x) ／ (5 * x + 2) = 7
3／ (4 * x * x − 5) = 10
(7 * x) ／ (12 * x * x + 1) = 3
(3 * x + 20) ／ (5 * x * x − 2) = 10
(2 * x * x + 5) ／ (10 * x * x + 2) = 20
(10 * x * x − 3 * x − 9) ／ (12 * x * x + 2) = 3
(4 * x * x + 3 * x) ／ (10 * x * x − 6) = 4
(3 * x * x) ／ (5 * x * x + 2) = 7
3／ (4 * x * x + 3 * x) = 10
(7 * x) ／ (12 * x * x − 2 * x) = 3
(3 * x + 20) ／ (5 * x * x − 13 * x) = 10
(2 * x * x + 5) ／ (10 * x * x + 5 * x) = 20
(10 * x * x − 3 * x − 9) ／ (12 * x * x + 6 * x) = 3
(4 * x * x + 3 * x) ／ (10 * x * x − 2 * x) = 4
(3 * x * x) ／ (5 * x * x − 8 * x) = 7
3／ (3 * x + 2) = 2 * x − 5
(7 * x) ／ (2 * x − 5) = 13 * x + 1
(3 * x + 20) ／ (13 * x + 1) = 5x − 20
(2 * x * x + 5) ／ (5x − 20) = 6 * x + 4
(10 * x * x − 3 * x − 9) ／ (6 * x + 4) = 2 * x − 12
(4 * x * x + 3 * x) ／ (2 * x − 12) = 8 * x + 7
(3 * x * x) ／ (8 * x + 7) = 3 * x + 2
3／x = 10

(7 * x) /x = 3
(3 * x + 20) /x = 10
(2 * x * x + 5) /x = 20
(10 * x * x − 3 * x − 9) /x = 3
(4 * x * x + 3 * x) /x = 4
(3 * x * x) /x = 7

対数方程式 : 78問

log (10,5 * x) = 2
log (5,3 * x − 4) = 1
log (4,3 * (4 * x − 2)) = 3
log (10,2 * x * x) = 2
log (3,4 * x * x − 1) = 5
log (4,8 * x * x + 2 * x) = 2
log (10,2 * x * x − 3 * x − 10) = 1
2 = log (10,2 * x)
log (3,2 * x + 6) = 4
2 = log (10,5 * x − 5)
log (10,6 * x + 3) = log (10,4 * x − 5)
log (10,5 * x) = log (10,2)
log (2,2 * x + 6) = log (2,34)
log (5,23) = log (5,5 * x − 5)
log (4,6 * x) + 3 = 7
log (7,2 * x) = 1
log (4,6 * x + 3) + 3 = 7
log (7,2 * x + 10) = 1
log (5,2 * x + 6) + log (5,18 * x) = 3
2 = log (10,5 * x)
log (10,2 * x) = log (10,20)
4 = log (3,(2 * x + 6))
log (10,(5 * x − 5)) = 2
log (10,(4 * x − 5)) = log (10,(6 * x + 3))
log (10,2) = log (10,5 * x)
log (2,34) = log (2,(2 * x + 6))
log (5,(5 * x − 5)) = log (5,23)
# 7 = log (4,6 * x) + 3
1 = log (7,2 * x)
# 7 = log (4,(6 * x + 3)) + 3
1 = log (7,(2 * x + 10))
# 3 = log (5,(2 * x + 6)) + log (5,18 * x)
1 = log (5,3 * x − 4)

```
   3 = log (4,3 * (4 * x - 2))
#  2 = log (10,2 * x * x)
#  5 = log (3,4 * x * x - 1)
   2 = log (4,8 * x * x + 2 * x)
#  1 = log (10,2 * x * x - 3 * x - 10)
#  log (2,3 * x) = log (2,5 * x)
#  log (2,3 * x + 1) = log (2,5 * x)
#  log (2,3 * x) = log (2,5 * x - 2)
   log (2,3 * x + 1) = log (2,5 * x - 10)
#  log (2,4 * x * x) = log (2,3 * x)
#  log (2,4 * x * x - 4 * x) = log (2,3 * x)
#  log (2,4 * x * x + 2) = log (2,3 * x)
#  log (2,4 * x * x - 3 * x + 10) = log (2,3 * x)
#  log (2,4 * x * x) = log (2,3 * x - 2)
#  log (2,4 * x * x - 4 * x) = log (2,3 * x - 2)
#  log (2,4 * x * x + 2) = log (2,3 * x - 2)
#  log (2,4 * x * x - 3 * x + 10) = log (2,3 * x - 2)
#  log (2,4 * x * x) = log (2,3 * x * x)
#  log (2,4 * x * x - 4 * x) = log (2,3 * x * x)
#  log (2,4 * x * x + 2) = log (2,3 * x * x)
#  log (2,4 * x * x - 3 * x + 10) = log (2,3 * x * x)
#  log (2,4 * x * x) = log (2,3 * x * x - 10)
#  log (2,4 * x * x - 4 * x) = log (2,3 * x * x - 10)
#  log (2,4 * x * x + 2) = log (2,3 * x * x - 10)
#  log (2,4 * x * x - 3 * x + 10) = log (2,3 * x * x - 10)
#  log (2,4 * x * x) = log (2,3 * x * x - 10 * x)
#  log (2,4 * x * x - 4 * x) = log (2,3 * x * x - 10 * x)
#  log (2,4 * x * x + 2) = log (2,3 * x * x - 10 * x)
#  log (2,4 * x * x - 3 * x + 10) = log (2,3 * x * x - 10 * x)
#  log (2,4 * x * x) = log (2,3 * x * x - 10 * x + 4)
#  log (2,4 * x * x - 4 * x) = log (2,3 * x * x - 10 * x + 4)
#  log (2,4 * x * x + 2) = log (2,3 * x * x - 10 * x + 4)
#  log (2,4 * x * x - 3 * x + 10) = log (2,3 * x * x - 10 * x + 4)
   log (4,6 * x) + 3 = 2
   log (4,6 * x - 2) + 3 = 4
#  log (2,6 * x * x) + 6 = 7
#  log (4,4 * x * x - 3) + 3 = 8
#  log (2,3 * x * x + 2 * x) + 9 = 7
#  log (2,3 * x * x + 10 * x - 1) + 3 = 2
   10 = log (4,6 * x) + 3
   3 = log (4,6 * x - 2) + 3
#  5 = log (2,6 * x * x) + 6
```

6 = log (4,4 * x * x − 3) + 3
4 = log (2,3 * x * x + 2 * x) + 9
2 = log (2,3 * x * x + 10 * x − 1) + 3

指数方程式 : 78問

exp (2,5 * x) = 2
exp (5,3 * x − 4) = 10
exp (4,3 * (4 * x − 2)) = 30
exp (10,2 * x * x) = 20
exp (3,4 * x * x − 1) = 50
exp (4,8 * x * x + 2 * x) = 20
exp (10,2 * x * x − 3 * x − 10) = 10
20 = exp (2,2 * x)
exp (3,2 * x + 6) = 40
20 = exp (10,5 * x − 5)
exp (10,6 * x + 3) = exp (10,4 * x − 5)
exp (10,5 * x) = exp (10,2)
exp (2,2 * x + 6) = exp (2,34)
exp (5,23) = exp (5,5 * x − 5)
exp (4,6 * x) + 3 = 70
exp (7,2 * x) = 10
exp (4,6 * x + 3) + 3 = 70
exp (7,2 * x + 10) = 10
# exp (5,2 * x + 6) * exp (5,18 * x) = 30
20 = exp (10,5 * x)
exp (10,2 * x) = exp (10,20)
40 = exp (3,(2 * x + 6))
exp (10,(5 * x − 5)) = 20
exp (10,(4 * x − 5)) = exp (10,(6 * x + 3))
exp (10,2) = exp (10,5 * x)
exp (2,34) = exp (2,(2 * x + 6))
exp (5,(5 * x − 5)) = exp (5,23)
# 70 = exp (4,6 * x) + 3
10 = exp (7,2 * x)
# 73 = exp (4,(6 * x + 3)) + 3
13 = exp (7,(2 * x + 10))
33 = exp (5,(2 * x + 6)) * exp (5,18 * x)
15 = exp (5,3 * x − 4)
35 = exp (4,3 * (4 * x − 2))
# 25 = exp (10,2 * x * x)
# 57 = exp (3,4 * x * x − 1)

```
# 27 = exp (4,8 * x * x + 2 * x)
# 10 = exp (10,2 * x * x - 3 * x - 10)
# exp (2,3 * x) = exp (2,5 * x)
# exp (2,3 * x + 1) = exp (2,5 * x)
# exp (2,3 * x) = exp (2,5 * x - 2)
  exp (2,3 * x + 1) = exp (2,5 * x - 10)
# exp (2,4 * x * x) = exp (2,3 * x)
# exp (2,4 * x * x - 4 * x) = exp (2,3 * x)
# exp (2,4 * x * x + 2) = exp (2,3 * x)
# exp (2,4 * x * x - 3 * x + 10) = exp (2,3 * x)
# exp (2,4 * x * x) = exp (2,3 * x - 2)
# exp (2,4 * x * x - 4 * x) = exp (2,3 * x - 2)
# exp (2,4 * x * x + 2) = exp (2,3 * x - 2)
# exp (2,4 * x * x - 3 * x + 10) = exp (2,3 * x - 2)
# exp (2,4 * x * x) = exp (2,3 * x * x)
# exp (2,4 * x * x - 4 * x) = exp (2,3 * x * x)
# exp (2,4 * x * x + 2) = exp (2,3 * x * x)
# exp (2,4 * x * x - 3 * x + 10) = exp (2,3 * x * x)
# exp (2,4 * x * x) = exp (2,3 * x * x - 10)
# exp (2,4 * x * x - 4 * x) = exp (2,3 * x * x - 10)
# exp (2,4 * x * x + 2) = exp (2,3 * x * x - 10)
# exp (2,4 * x * x - 3 * x + 10) = exp (2,3 * x * x - 10)
# exp (2,4 * x * x) = exp (2,3 * x * x - 10 * x)
# exp (2,4 * x * x - 4 * x) = exp (2,3 * x * x - 10 * x)
# exp (2,4 * x * x + 2) = exp (2,3 * x * x - 10 * x)
# exp (2,4 * x * x - 3 * x + 10) = exp (2,3 * x * x - 10 * x)
# exp (2,4 * x * x) = exp (2,3 * x * x - 10 * x + 4)
# exp (2,4 * x * x - 4 * x) = exp (2,3 * x * x - 10 * x + 4)
# exp (2,4 * x * x + 2) = exp (2,3 * x * x - 10 * x + 4)
# exp (2,4 * x * x - 3 * x + 10) = exp (2,3 * x * x - 10 * x + 4)
  exp (4,6 * x) + 3 = 29
  exp (4,6 * x - 2) + 3 = 48
# exp (2,6 * x * x) + 6 = 77
# exp (4,4 * x * x - 3) + 3 = 86
# exp (2,3 * x * x + 2 * x) + 9 = 78
# exp (2,3 * x * x + 10 * x - 1) + 3 = 29
  100 = exp (4,6 * x) + 3
  33 = exp (4,6 * x - 2) + 3
# 52 = exp (2,6 * x * x) + 6
# 64 = exp (4,4 * x * x - 3) + 3
# 46 = exp (2,3 * x * x + 2 * x) + 9
# 29 = exp (2,3 * x * x + 10 * x - 1) + 3
```

< Appendix3 > 獲得されたマクロオペレータと絶対オペレータ

[2次方程式]

マクロオペレータ ：62

rule200 $ 0__94 : (AL * AL + R1) * R2 → R2 * AL * AL + R3
rule351 $ 20__92 : (R1 * AL * AL + R2 * AL + R3) /NR1 → R4 * AL * AL + R5 * AL + R6
rule342 $ 63__93 : (R1 * AL * AL) /NR1 → R2 * AL * AL
rule010 $ 2__91 : R1 * AL = R2 * AL → 1 * AL = R3 (DS)
rule000 $ 3__90 : R1 * AL = NR1 * AL * AL → 1 * AL = R2 (DS)
rule000 $ 3__89 : R1 = NR1 * AL * AL → AL = R2 (DS)
rule010 $ 2__88 : R1 * AL * AL = R2 * AL * AL → AL = R3 (DS)
rule010 $ 2__87 : R1 * AL * AL + R2 = R3 * AL * AL → AL = R4 (DS)
rule010 $ 2__86 : NR1 * AL * AL = R1 * AL + R2 → 1 * AL = R3 (DS)
rule010 $ 2__85 : NR1 * AL * AL + R1 = R2 * AL + R3 → 1 * AL = R4 (DS)
rule000 $ 3__84 : R1 * AL = NR1 * AL * AL + R2 → 1 * AL = R3 (DS)
rule000 $ 3__83 : R1 = NR1 * AL * AL + R2 → AL = R3 (DS)
rule010 $ 2__82 : R1 * AL * AL = R2 * AL * AL + R3 → AL = R4 (DS)
rule010 $ 2__81 : R1 * AL + R2 = R3 * AL * AL + R4 → 1 * AL = R5 (DS)
rule010 $ 2__80 : R1 * AL * AL + R2 = R3 * AL * AL + R4 → AL = R5 (DS)
rule010 $ 2__79 : R1 * AL * AL + R2 * AL = R3 * AL * AL + R4 → 1 * AL = R5 (DS)
rule000 $ 3__77 : R1 * AL = NR1 * AL * AL + R2 * AL → 1 * AL = R3 (DS)
rule010 $ 7__78 : NR1 * AL * AL + R1 * AL = R2 * AL → 1 * AL = R3 (DS)
rule000 $ 3__76 : R1 = NR1 * AL * AL + R2 * AL → 1 * AL = R3 (DS)
rule000 $ 3__74 : R1 * AL * AL = R2 * AL * AL + R3 * AL → 1 * AL = R4 (DS)
rule010 $ 7__75 : R1 * AL * AL + R2 * AL = R3 * AL * AL → 1 * AL = R4 (DS)
rule010 $ 2__73 : R1 * AL + R2 = R3 * AL * AL + R4 * AL → 1 * AL = R5 (DS)
rule010 $ 2__71 : R1 * AL * AL + R2 = R3 * AL * AL + NR1 * AL → 1 * AL = R4 (DS)
rule010 $ 10__72 : NR1 * AL * AL + R1 = R2 * AL → 1 * AL = R3 (DS)
rule010 $ 2__70 : R1 * AL * AL + R2 * AL = R3 * AL * AL + R4 * AL → 1 * AL = R5 (DS)
rule010 $ 2__69 : R1*AL * AL + R2 * AL + R3 = R4 * AL * AL + R5 * AL → 1 * AL = R6 (DS)
rule000 $ 3__67 : R1 * AL = NR1 * AL * AL + R2 * AL + R3 → 1 * AL = R4 (DS)
rule010 $ 7__68 : NR1 * AL * AL + R1 * AL + R2 = R3 * AL → 1 * AL = R4 (DS)
rule000 $ 3__66 : R1 = NR1 * AL * AL + R2 * AL + R3 → 1 * AL = R4 (DS)
rule000 $ 3__64 : R1 * AL * AL = R2 * AL * AL + R3 * AL + R4 → 1 * AL = R5 (DS)
rule010 $ 7__65 : R1 * AL * AL + R2 * AL + R3 = R4 * AL * AL → 1 * AL = R5 (DS)
rule000 $ 3__63 : R1 * AL + R2 = NR1 * AL * AL + R3 * AL + R4 → 1 * AL = R5 (DS)
rule000 $ 3__61 : R1 * AL * AL + R2 = R3 * AL * AL + R4 * AL + R5 → 1 * AL = R6 (DS)
rule010 $ 7__62 : R1 * AL * AL + R2 * AL + R3 = R4 * AL * AL + R5 → 1 * AL = R6 (DS)

rule010 $ 2__59 : R1*AL* AL + R2 * AL = R3 * AL * AL + R4 * AL + R5 → 1 * AL = R6 (DS)
rule010 $ 10__60 : NR1 * AL * AL + R1 * AL = R2 * AL + R3 → 1 * AL = R4 (DS)
rule211__5 $ 7__57 : R1 * AL * AL + R2 * AL * AL → NR1 * AL * AL
rule010 $ 2__56 : R1*AL*AL + R2 *AL + R3 = R4 *AL*AL + R5*AL + R6
→ 1*AL = R7 (DS)
rule010 $ 14__58 : NR1 * AL * AL + R1 * AL + R2 = R3 * AL + R4 → 1 * AL = R5 (DS)
rule352 $ 0__55 : (R1 * AL * AL + R2 * AL + R3) * R4 → NR1 * AL * AL + R5 * AL + R6
rule211 $ 5__54 : R1 * A * A − R1 * A * A → 0
rule010 $ 2__53 : R1 * AL + R2 = R3 * AL * AL → 1 * AL = R4 (DS)
rule010 $ 2__52 : AL * AL = R1 * AL + R2 → 1 * AL = R3 (DS)
rule010 $ 2__51 : AL * AL + R1 = R2 * AL → 1 * AL = R3 (DS)
rule010 $ 2__50 : AL * AL = R1 * AL → 1 * AL = R2 (DS)
rule010 $ 2__49 : NR1 * AL * AL = R1 * AL → 1 * AL = R2 (DS)
rule350 $ 6__48 : (NR1 * AL * AL + R1 * AL) /NR1 → AL * AL + R2 * AL
rule030 $ 0__47 : NR1 * AL * AL + R1 * AL = R2 → 1 * AL = R3 (DS)
rule200__11 $ 0__46 : (1 * AL + R1) * AL → AL * AL + R1 * AL
rule200 $ 0__44 : (R1 * AL + R2) * (R3 * AL + R4) → NR1 * AL * AL + R5 * AL + R6
rule210__16 $ 9__45 : AL * R1 + AL * (NR1 * AL + R2) → NR1 * AL * AL + R3 * AL
rule351 $ 6__43 : (NR1 * AL * AL + R1 * AL + R2) /NR1 → AL * AL + R3 * AL + R4
rule030 $ 0__42 : NR1 * AL * AL + R1 * AL + R2 = R3 → 1 * AL = R4 (DS)
rule010 $ 3__41 : AL * AL + R1 * AL + R2 = R3 → 1 * AL = R4 (DS)
rule430 $ 37__40 : R1 * AL → 2 * R2 * AL
rule010 $ 3__39 : AL * AL + R1 * AL = R2 → 1 * AL = R3 (DS)
rule010 $ 6__38 : (NR1 * AL + R1) * (NR1 * AL + R1) + R2 = R3 → 1 * AL = R4 (DS)
rule440 $ 0__37 : (NR1 * AL + R1) * (NR1 * AL + R1) = A → 1 * AL = R2 (DS)
rule010 $ 2__36 : NR1 * AL * AL + R1 = R2 → AL = R3 (DS)
rule342 $ 7__35 : (NA * AL * AL) /NA → AL * AL
rule030 $ 0__34 : NR1 * AL * AL = R1 → AL = R2 (DS)
rule010 $ 2__33 : AL * AL + R1 = R2 → AL = R3 (DS)

[分数方程式]

マクロオペレータ ：58

rule343 $ 222__152 : (R1 * NA * AL) /NA → NR1 * AL
rule020 $ 221__150 : (NR1 * AL * AL + R1 * AL) /AL = R2 → 1 * AL = R3 (DS)
rule345__126 $ 225__151 : AL * ((NR1 * AL * AL + R1 * AL) /AL)
→ NR2 * AL * AL + R2 * AL
rule020 $ 219__148 : (R1 * AL * AL + R2 * AL + R3) /AL = R4 → 1 * AL = R5 (DS)
rule345__119 $ 223__149 : AL * ((R1 * AL * AL + R2 * AL + R3) /AL)
→ NR1 * AL * AL + R4 * AL + R5

rule020 $ 216__146 : (NR1 * AL * AL + R1) ／AL = R2 → 1 * AL = R3 (DS)
rule345__100 $ 220__147 : AL * ((NR1 * AL * AL + R1) ／AL) → NR2 * AL * AL + R2
rule020 $ 214__144 : (R1 * AL + R2) ／AL = R3 → 1 * AL = R4 (DS)
rule345__93 $ 218__145 : AL * ((R1 * AL + R2) ／AL) → NR1 * AL + R3
rule020 $ 212__142 : (R1 * AL) ／AL = R2 → 1 * AL = R3 (DS)
rule345__86 $ 216__143 : AL * ((R1 * AL) ／AL) → R2 * AL
rule020 $ 210__140 : R1／AL = NR1 → 1 * AL = R2 (DS)
rule345__77 $ 214__141 : NA * (R1／NA) → R2
rule020 $ 208__139 : (R1 * AL * AL) ／ (R2 * AL + R3) = R4 * AL + R5 → 1 * AL = R6 (DS)
rule020 $ 204__138 : (NR1 * AL * AL + R1 * AL) ／ (R2 * AL + R3) = R4 * AL + R5
→ 1 * AL = R6 (DS)
rule020 $ 202__137 : (R1 * AL * AL + R2 * AL + R3) ／ (R4 * AL + R5) = R6 * AL + R7
→ 1 * AL = R8 (DS)
rule020 $ 199__136 : (NR1 * AL * AL + R1) ／ (R2 * AL + R3) = R4 * AL + R5
→ 1 * AL = R6 (DS)
rule020 $ 197__135 : (R1 * AL + R2) ／ (R3 * AL + R4) = R5 * AL + R6 → 1 * AL = R7 (DS)
rule020 $ 195__134 : (R1 * AL) ／ (R2 * AL + R3) = R4 * AL + R5 → 1 * AL = R6 (DS)
rule020 $ 193__133 : R1／ (R2 * AL + R3) = R4 * AL + R5 → 1 * AL = R6 (DS)
rule020 $ 191__132 : (R1 * AL * AL)／ (R2 * AL * AL + R3 * AL) = R4 → 1 * AL = R5 (DS)
rule020 $ 187__130 : (NR1 * AL * AL + R1 * AL)／(R2 * AL * AL + R3 * AL) = R4
→ 1 * AL = R5 (DS)
rule345__126 $ 192__131 : (A*AL*AL+B*AL)*((NR1*AL*AL+ R1*AL)／(A*AL*AL+B
*AL)) → NR2 * AL * AL + R2 * AL
rule020 $ 185__129 : (R1 * AL * AL + R2 * AL + R3) ／ (R4 * AL * AL + R5 * AL) = R6
→ 1 * AL = R7 (DS)
rule020 $ 182__128 : (NR1 * AL * AL + R1) ／ (R2 * AL * AL + R3 * AL) = R4
→ 1 * AL = R5 (DS)
rule020 $ 180__127 : (R1 * AL + R2) ／ (R3 * AL * AL + R4 * AL) = R5 → 1 * AL = R6 (DS)
rule020 $ 178__126 : (R1 * AL) ／ (R2 * AL * AL + R3 * AL) = R4 → 1 * AL = R5 (DS)
rule020 $ 171__124 : R1／ (R2 * AL * AL + R3 * AL) = R4 → 1 * AL = R5 (DS)
rule200 $ 176__125 : (R1 * AL * AL + R2 * AL) * R3 → NR1 * AL * AL + R4 * AL
rule020 $ 169__122 : (R1 * AL * AL) ／ (R2 * AL * AL + R3) = R4 → AL = R5 (DS)
rule345__145 $ 174__123 : (A * AL * AL + B) * ((R1 * AL * AL) ／ (A * AL * AL + B))
→ R2 * AL * AL
rule020 $ 165__120 : (R1 * AL * AL + R2 * AL)／(R3 * AL * AL + R4) = R5
→ 1 * AL = R6 (DS)
rule345__126 $ 170__121 : (A *AL*AL + B)*((R1 * AL * AL + R2 * AL)／(A * AL * AL + B))
→ R1 * AL * AL + R2 * AL
rule020 $ 163__118 : (R1 * AL * AL + R2 * AL + R3) ／ (R4 * AL * AL + R5) = R6
→ 1 * AL = R7 (DS)
rule345__119 $ 168__119 : (A* AL*AL+B)*((R1* AL* AL+R2*AL+R3)／(A*AL*AL+B))
→ NR1 * AL * AL + R4 * AL + R5

rule020 $ 160__116 : (NR1 * AL * AL + R1) ／ (R2 * AL * AL + R3) = R4 → AL = R5 (DS)
rule345__100 $ 165__117 : (A*AL*AL + B)*((NR1*AL*AL + R1)／(A* AL * AL + B))
→ NR2 * AL * AL + R2
rule020 $ 158__114 : (R1 * AL + R2) ／ (R3 * AL * AL + R4) = R5 → 1 * AL = R6 (DS)
rule345__93 $ 163__115 : (A * AL * AL + B) * ((R1 * AL + R2) ／ (A * AL * AL + B))
→ NR1 * AL + R3
rule020 $ 156__112 : (R1 * AL) ／ (R2 * AL * AL + R3) = R4 → 1 * AL = R5 (DS)
rule345__86 $ 161__113 : (A * AL * AL + B)*((R1 * AL)／(A * AL * AL + B)) → R2 * AL
rule020 $ 147__109 : R1／ (R2 * AL * AL + R3) = R4 → AL = R5 (DS)
rule345__77 $ 151__110 : (A * B * B + C) * (R1／ (A * B * B + C)) → R2
rule200 $ 154__111 : (R1 * AL * AL + R2) * R3 → NR1 * AL * AL + R4
rule020 $ 140__107 : (R1 * AL * AL) ／ (R2 * AL + R3) = R4 → 1 * AL = R5 (DS)
rule345 $ 145__108 : (A * AL + B) * ((R1 * AL * AL)／(A * AL + B)) → NR1 * AL * AL
rule020 $ 121__105 : (NR1 * AL * AL + R1 * AL)／(R2 * AL + R3) = R4 → 1 * AL = R5 (DS)
rule345 $ 126__106 : (A * AL + B) * ((NR1 * AL * AL + R1 * AL) ／ (A * AL + B))
→ NR1 * AL * AL + R1 * AL
rule020 $ 113__103 : (R1 * AL * AL + R2 * AL + R3)／(R4 *AL + R5)= R6 → 1*AL = R7 (DS)
rule345 $ 119__104 : (A * AL + B) * ((R1 * AL * AL + R2 * AL + R3) ／ (A * AL + B))
→ NR1 * AL * AL + R4 * AL + R5
rule020 $ 95__101 : (NR1 * AL * AL + R1) ／ (R2 * AL + R3) = R4 → 1 * AL = R5 (DS)
rule345 $ 100__102 : (A*AL+B)*((NR1*AL* AL + R1)／(A*AL + B)) → NR1*AL * AL + R1
rule020 $ 88__99 : (R1 * AL + R2) ／ (R3 * AL + R4) = R5 → 1 * AL = R6 (DS)
rule345 $ 93__100 : (A * AL + B) * ((R1 * AL + R2) ／ (A * AL + B)) → NR1 * AL + R3
rule020 $ 81__97 : (R1 * AL) ／ (R2 * AL + R3) = R4 → 1 * AL = R5 (DS)
rule345 $ 86__98 : (A * AL + B) * ((R1 * AL) ／ (A * AL + B)) → R2 * AL
rule020 $ 72__95 : R1／ (R2 * AL + R3) = R4 → 1 * AL = R5 (DS)
rule345 $ 77__96 : (A * B + C) * (R1／ (A * B + C)) → R2

[対数方程式]

マクロオペレータ ：56

rule000 $ 251__150 : R1 = log (R2,(NR1*AL * AL + R3 * AL + R4)) + R5 → 1 * AL = R6 (DS)
rule000 $ 247__149 : R1 = log (R2,(NR1 * AL * AL + R3 * AL)) + R4 → 1 * AL = R5 (DS)
rule000 $ 243__148 : R1 = log (R2,(NR1 * AL * AL + R3)) + R4 → AL = R5 (DS)
rule000 $ 239__147 : R1 = log (R2,NR1 * AL * AL) + R3 → AL = R4 (DS)
rule010 $ 229__146 : log (R1,(NR1 * AL * AL + R2 * AL + R3))+ R4 = R5 → 1 * AL = R6 (DS)
rule010 $ 220__145 : log (R1,(NR1 * AL * AL + R2 * AL)) + R3 = R4 → 1 * AL = R5 (DS)
rule010 $ 211__144 : log (R1,(NR1 * AL * AL + R2)) + R3 = R4 → AL = R5 (DS)
rule010 $ 202__143 : log (R1,NR1 * AL * AL) + R2 = R3 → AL = R4 (DS)
rule110 $ 199__142 : log(A,(R1*AL*AL+ R2 * AL+R3))= log(A,(R4*AL*AL+ R5 * AL + R6))

→ 1 * AL = R7　(DS)

rule110 $ 198__141 : log (A,(R1 * AL * AL + R2)) = log (A,(R3 * AL * AL + R4 * AL + R5))
→ 1 * AL = R6　(DS)

rule110 $ 197__140 : log(A,(R1*AL* AL+ R2 * AL)) = log (A,(R3 * AL * AL + R4 * AL + R5))
→ 1 * AL = R6　(DS)

rule110 $ 196__139 : log (A,R1 * AL * AL) = log (A,(R2 * AL * AL + R3 * AL + R4))
→ 1 * AL = R5　(DS)

rule110 $ 195__138 : log(A,(R1 *AL*AL + R2* AL + R3)) = log (A,(R4 * AL * AL + R5 * AL))
→ 1 * AL = R6　(DS)

rule110 $ 193__137 : log (A,(R1 * AL * AL + R2)) = log (A,(R3 * AL * AL + NR1 * AL))
→ 1 * AL = R4　(DS)

rule110 $ 191__136 : log (A,(R1 * AL * AL + R2 * AL)) = log (A,(R3 * AL * AL + R4 * AL))
→ 1 * AL = R5　(DS)

rule110 $ 189__135 : log (A,R1 * AL * AL) = log (A,(R2 * AL * AL + R3 * AL))
→ 1 * AL = R4　(DS)

rule110 $ 187__134 : log (A,(R1 * AL * AL + R2 * AL + R3)) = log (A,(R4 * AL * AL + R5))
→ 1 * AL = R6　(DS)

rule110 $ 185__133 : log (A,(R1 * AL * AL + R2)) = log (A,(R3 * AL * AL + R4))
→ AL = R5　(DS)

rule110 $ 183__132 : log (A,(R1 * AL * AL + R2 * AL)) = log (A,(R3 * AL * AL + R4))
→ 1 * AL = R5　(DS)

rule110 $ 181__131 : log (A,R1 * AL * AL) = log (A,(R2 * AL * AL + R3)) → AL = R4　(DS)

rule110 $ 179__130 : log (A,(R1 * AL * AL + R2 * AL + R3)) = log (A,R4 * AL * AL)
→ 1 * AL = R5　(DS)

rule110 $ 177__129 : log (A,(R1 * AL * AL + R2 * AL)) = log (A,R3 * AL * AL)
→ 1 * AL = R4　(DS)

rule110 $ 175__128 : log (A,R1 * AL * AL) = log (A,R2 * AL * AL) → AL = R3　(DS)

rule110 $ 173__127 : log (A,(NR1 * AL * AL + R1 * AL + R2)) = log (A,(R3 * AL + R4))
→ 1 * AL = R5　(DS)

rule110 $ 171__126 : log(A,(NR1*AL*AL + R1)) = log(A,(R2 * AL + R3)) → 1 * AL = R4　(DS)

rule110 $ 169__125 : log (A,(NR1 * AL * AL + R1 * AL)) = log (A,(R2 * AL + R3))
→ 1 * AL = R4　(DS)

rule110 $ 167__124 : log (A,NR1 * AL * AL) = log (A,(R1 * AL + R2)) → 1 * AL = R3　(DS)

rule110 $ 165__123 : log(A,(NR1*AL*AL+ R1*AL+ R2)) = log(A,R3* AL) → 1*AL = R4　(DS)

rule110 $ 163__122 : log (A,(NR1 * AL * AL + R1)) = log (A,R2 * AL) → 1 * AL = R3　(DS)

rule110 $ 161__121 : log (A,(NR1* AL * AL + R1 * AL)) = log(A,R2 * AL) → 1*AL = R3　(DS)

rule110 $ 159__120 : log (A,NR1 * AL * AL) = log (A,R1 * AL) → 1 * AL = R2　(DS)

rule110 $ 157__119 : log (A,R1 * AL) = log (A,(R2 * AL + R3)) → 1 * AL = R4　(DS)

rule110 $ 155__118 : log (A,(R1 * AL + R2)) = log (A,R3 * AL) → 1 * AL = R4　(DS)

rule110 $ 153__117 : log (A,R1 * AL) = log (A,R2 * AL) → 1 * AL = R3　(DS)

rule000 $ 151__116 : R1 = log (R2,(NR1 * AL * AL + R3 * AL + R4)) → 1 * AL = R5　(DS)

rule000 $ 147__115 : R1 = log (R2,(NR1 * AL * AL + R3)) → AL = R4　(DS)

rule000 $ 143__114 : R1 = log (R2,NR1 * AL * AL) → AL = R3 (DS)
rule000 $ 139__113 : R1 = log (R2,(NR1 * AL * AL + R3 * AL)) → 1 * AL = R4 (DS)
rule000 $ 134__112 : R1 = log (R2,(NR1 * AL + R3)) + R4 → 1 * AL = R5 (DS)
rule000 $ 130__111 : R1 = log (R2,NR1 * AL) + R3 → 1 * AL = R4 (DS)
rule110 $ 125__110 : log (A,R1) = log (A,NR1 * AL) → 1 * AL = R2 (DS)
rule103 $ 123__109 : log (R4,R1 * AL) + log (R4,(R2 * AL + R3))
→ log (R4,(NR1 * AL * AL + R5 * AL))
rule010 $ 116__108 : log (R1,(NR1 * AL + R2)) + R3 = R4 → 1 * AL = R5 (DS)
rule010 $ 107__107 : log (R1,NR1 * AL) + R2 = R3 → 1 * AL = R4 (DS)
rule110 $ 103__106 : log (A,R1) = log (A,(NR1 * AL + R2)) → 1 * AL = R3 (DS)
rule110 $ 101__105 : log (A,(NR1 * AL + R1)) = log (A,R2) → 1 * AL = R3 (DS)
rule110 $ 99__104 : log (A,NR1 * AL) = log (A,R1) → 1 * AL = R2 (DS)
rule110 $ 97__103 : log (A,(NR1 * AL + R1)) = log (A,(R2 * AL + R3)) → 1 * AL = R4 (DS)
rule000 $ 95__102 : R1 = log (R2,(NR1 * AL + R3)) → 1 * AL = R4 (DS)
rule000 $ 90__101 : R1 = log (R2,NR1 * AL) → 1 * AL = R3 (DS)
rule111 $ 84__100 : log (R1,(NR1 * AL * AL + R2 * AL + R3)) = R4 → 1 * AL = R5 (DS)
rule111 $ 83__99 : log (R1,(NR1 * AL * AL + R2 * AL)) = R3 → 1 * AL = R4 (DS)
rule111 $ 82__98 : log (R1,(NR1 * AL * AL + R2)) = R3 → AL = R4 (DS)
rule111 $ 81__97 : log (R1,NR1 * AL * AL) = R2 → AL = R3 (DS)
rule111 $ 80__96 : log (R1,(NR1 * AL + R2)) = R3 → 1 * AL = R4 (DS)
rule111 $ 71__95 : log (R1,NR1 * AL) = R2 → 1 * AL = R3 (DS)

絶対オペレータ ：1

rule127__79 : exp (R1,R2) → R3

[指数方程式]

マクロオペレータ ：57

rule000 $ 225__151 : R1 = exp(R2,(NR1 * AL * AL + R3 * AL + R4)) + R5 → 1 * AL = R6 (DS)
rule000 $ 221__150 : R1 = exp (R2,(NR1 * AL * AL + R3 * AL)) + R4 → 1 * AL = R5 (DS)
rule000 $ 217__149 : R1 = exp (R2,(NR1 * AL * AL + R3)) + R4 → AL = R5 (DS)
rule000 $ 213__148 : R1 = exp (R2,NR1 * AL * AL) + R3 → AL = R4 (DS)
rule010 $ 203__147 : exp(R1,(NR1* AL * AL + R2 * AL + R3)) + R4 = R5 → 1 * AL = R6 (DS)
rule010 $ 194__146 : exp (R1,(NR1 * AL * AL + R2 * AL)) + R3 = R4 → 1 * AL = R5 (DS)
rule010 $ 185__145 : exp (R1,(NR1 * AL * AL + R2)) + R3 = R4 → AL = R5 (DS)
rule010 $ 176__144 : exp (R1,NR1 * AL * AL) + R2 = R3 → AL = R4 (DS)
rule126 $ 173__143 : exp(A,(R1*AL*AL + R2*AL + R3)) = exp(A,(R4*AL*AL+R5*AL+ R6))
→ 1 * AL = R7 (DS)
rule126 $ 172__142 : exp (A,(R1 * AL * AL + R2)) = exp (A,(R3 * AL * AL + R4 * AL + R5))

→ 1 * AL = R6 (DS)

rule126 $ 171__141 : exp (A,(R1*AL*AL + R2*AL)) = exp (A,(R3*AL*AL+ R4*AL + R5))
→ 1 * AL = R6 (DS)

rule126 $ 170__140 : exp (A,R1 * AL * AL) = exp (A,(R2 * AL * AL + R3 * AL + R4))
→ 1 * AL = R5 (DS)

rule126 $ 169__139 : exp(A,(R1*AL*AL + R2 * AL + R3)) = exp(A,(R4*AL * AL + R5 * AL))
→ 1 * AL = R6 (DS)

rule126 $ 168__138 : exp (A,(R1 * AL * AL + R2)) = exp (A,(R3 * AL * AL + NR1 * AL))
→ 1 * AL = R4 (DS)

rule126 $ 167__137 : exp (A,(R1 * AL * AL + R2 * AL)) = exp (A,(R3 * AL * AL + R4 * AL))
→ 1 * AL = R5 (DS)

rule126 $ 166__136 : exp (A,R1 * AL * AL) = exp (A,(R2 * AL * AL + R3 * AL))
→ 1 * AL = R4 (DS)

rule126 $ 165__135 : exp (A,(R1 * AL * AL + R2 * AL + R3)) = exp (A,(R4 * AL * AL + R5))
→ 1 * AL = R6 (DS)

rule126 $ 164__134 : exp (A,(R1 * AL * AL + R2)) = exp (A,(R3 * AL * AL + R4))
→ AL = R5 (DS)

rule126 $ 163__133 : exp (A,(R1 * AL * AL + R2 * AL)) = exp (A,(R3 * AL * AL + R4))
→ 1 * AL = R5 (DS)

rule126 $ 162__132 : exp (A,R1 * AL * AL) = exp (A,(R2 * AL * AL + R3)) → AL = R4 (DS)

rule126 $ 161__131 : exp (A,(R1 * AL * AL + R2 * AL + R3)) = exp (A,R4 * AL * AL)
→ 1 * AL = R5 (DS)

rule126 $ 160__130 : exp (A,(R1 * AL * AL + R2)) = exp (A,R3 * AL * AL) → AL = R4 (DS)

rule126 $ 159__129 : exp (A,(R1 * AL * AL + R2 * AL)) = exp (A,R3 * AL * AL)
→ 1 * AL = R4 (DS)

rule126 $ 158__128 : exp (A,R1 * AL * AL) = exp (A,R2 * AL * AL) → AL = R3 (DS)

rule126 $ 157__127 : exp (A,(NR1 * AL * AL + R1 * AL + R2)) = exp (A,(R3 * AL + R4))
→ 1 * AL = R5 (DS)

rule126 $ 156__126 : exp (A,(NR1 * AL * AL + R1)) = exp (A,(R2 * AL + R3))
→ 1 * AL = R4 (DS)

rule126 $ 155__125 : exp (A,(NR1 * AL * AL + R1 * AL)) = exp (A,(R2 * AL + R3))
→ 1 * AL = R4 (DS)

rule126 $ 154__124 : exp (A,NR1 * AL * AL) = exp (A,(R1 * AL + R2)) → 1 * AL = R3 (DS)

rule126 $ 153__123 : exp (A,(NR1 * AL * AL + R1 * AL + R2)) = exp (A,R3 * AL)
→ 1 * AL = R4 (DS)

rule126 $ 152__122 : exp (A,(NR1 * AL * AL + R1)) = exp (A,R2 * AL) → 1 * AL = R3 (DS)

rule126 $ 151__121 : exp (A,(NR1 * AL * AL + R1 * AL)) = exp (A,R2 * AL)
→ 1 * AL = R3 (DS)

rule126 $ 150__120 : exp (A,NR1 * AL * AL) = exp (A,R1 * AL) → 1 * AL = R2 (DS)

rule126 $ 149__119 : exp (A,R1 * AL) = exp (A,(R2 * AL + R3)) → 1 * AL = R4 (DS)

rule126 $ 148__118 : exp (A,(R1 * AL + R2)) = exp (A,R3 * AL) → 1 * AL = R4 (DS)

rule126 $ 147__117 : exp (A,R1 * AL) = exp (A,R2 * AL) → 1 * AL = R3 (DS)

rule000 $ 146__116 : R1 = exp (R2,(NR1 * AL * AL + R3 * AL + R4)) → 1 * AL = R5 (DS)
rule000 $ 142__115 : R1 = exp (R2,(NR1 * AL * AL + R3 * AL)) → 1 * AL = R4 (DS)
rule000 $ 138__114 : R1 = exp (R2,(NR1 * AL * AL + R3)) → AL = R4 (DS)
rule000 $ 134__113 : R1 = exp (R2,NR1 * AL * AL) → AL = R3 (DS)
rule000 $ 130__112 : R1 = exp (R2,(NR1 * AL + R3)) + R4 → 1 * AL = R5 (DS)
rule000 $ 126__111 : R1 = exp (R2,NR1 * AL) + R3 → 1 * AL = R4 (DS)
rule000 $ 122__110 : exp (A,R1) = exp (A,NR1 * AL) → 1 * AL = R2 (DS)
rule121 $ 116__109 : (exp (R3,R1 * AL)) * (exp (R3,(R2 * AL + R4)))
→ exp (R3,(NR1 * AL + R4))
rule010 $ 109__108 : exp (R1,(NR1 * AL + R2)) + R3 = R4 → 1 * AL = R5 (DS)
rule010 $ 100__107 : exp (R1,NR1 * AL) + R2 = R3 → 1 * AL = R4 (DS)
rule126 $ 97__106 : exp (A,R1) = exp (A,(NR1 * AL + R2)) → 1 * AL = R3 (DS)
rule126 $ 95__105 : exp (A,(NR1 * AL + R1)) = exp (A,R2) → 1 * AL = R3 (DS)
rule126 $ 93__104 : exp (A,NR1 * AL) = exp (A,R1) → 1 * AL = R2 (DS)
rule126 $ 91__103 : exp (A,(NR1 * AL + R1)) = exp (A,(R2 * AL + R3)) → 1 * AL = R4 (DS)
rule000 $ 90__102 : R1 = exp (R2,(NR1 * AL + R3)) → 1 * AL = R4 (DS)
rule000 $ 86__101 : R1 = exp (R2,NR1 * AL) → 1 * AL = R3 (DS)
rule128 $ 82__100 : exp (R1,(NR1 * AL * AL + R2 * AL + R3)) = R4 → 1 * AL = R5 (DS)
rule128 $ 81__99 : exp (R1,(NR1 * AL * AL + R2 * AL)) = R3 → 1 * AL = R4 (DS)
rule128 $ 80__98 : exp (R1,(NR1 * AL * AL + R2)) = R3 → AL = R4 (DS)
rule128 $ 79__97 : exp (R1,NR1 * AL * AL) = R2 → AL = R3 (DS)
rule128 $ 78__96 : exp (R1,(NR1 * AL + R2)) = R3 → 1 * AL = R4 (DS)
rule128 $ 71__95 : exp (R1,NR1 * AL) = R2 → 1 * AL = R3 (DS)

絶対オペレータ ：1

rule112__77 : log (R1,R2) → R3

＜Appendix 4＞ DSBGの必須訓練例

1次方程式：6

5＊x＝（－5）
10＝2＊x
2＊x＋6＝12
6＊x＋3＝4＊x－5
3＊（4＊x－2）＝5＊x＋8
(12＊x－6）／3＋2＊x＝9

2次方程式：30

x＊x－9＝0
2＊x＊x＝8
(1＊x＋3）＊（1＊x＋3）＝16
x＊x＋6＊x＝5
x＊x－4＊x＋2＝0
2＊x＊x＋10＊x＋6＝0
(4＊x－3）＊（4＊x－2）＝0
2＊x＊x＋10＊x＝0
3＊x＊x＝（－2）＊x
x＊x＝6＊x
x＊x＋36＝（－12）＊x
x＊x＝（－6）＊x－9
(－20）＊y＋100＝2＊y＊y
5＊（3＊x＊x－2＊x＋9）＝3
10＊x＊x－3＊x－9＝4＊x＊x－36＊x＋6
10＊x＊x－3＊x＝4＊x＊x－36＊x＋6
10＊x＊x－9＝4＊x＊x－36＊x＋6
10＊x＊x＝4＊x＊x－36＊x＋6
10＊x＊x＝4＊x＊x－36＊x＋6
(－3）＊x＝4＊x＊x－36＊x＋6
10＊x＊x－3＊x＝4＊x＊x－36＊x
10＊x＊x＝4＊x＊x－36＊x
(－3）＊x＝4＊x＊x－36＊x
10＊x＊x－9＝4＊x＊x＋6
10＊x＊x＝4＊x＊x＋6
(－3）＊x＝4＊x＊x＋6
10＊x＊x＝4＊x＊x
(－3）＊x＝（－36）＊x

((4＊x＋2) ＊ (1＊x＋3)) ／2＝1＊x＊x
4＊ (x＊x－2) ＝3＊x＊x－2＊x

分数方程式：5

3／ (4＊x－5) ＝10
3／ (4＊x＊x－5) ＝10
3／ (4＊x＊x＋3＊x) ＝10
3／ (3＊x＋2) ＝2＊x－5
3／x＝10

対数方程式：3

log (10,5＊x) ＝2
log (10,(6＊x＋3)) ＝log (10,(4＊x－5))
log (5,18＊x) ＋log (5,(2＊x＋6)) ＝3

指数方程式：3

exp (2,5＊x) ＝2
exp (10,(6＊x＋3)) ＝exp (10,(4＊x－5))
(exp (5,18＊x)) ＊ (exp (5,(2＊x＋6))) ＝30

＜Appendix5＞ DSBGにおいて獲得された戦略知識

[1次方程式]

マクロオペレータ

rule000 $ 3__22__dsbg7 : R1 * AL + R2 = NR1 * AL + R3 → 1 * AL = R4 (DS)
rule000 $ 3__22__dsbg8 : R1 * AL + R2 = R3 * AL → 1 * AL = R4 (DS)
rule000 $ 3__22__dsbg4 : R1 = NR1 * AL + R2 → 1 * AL = R3 (DS)

rule350 $ 51__68 : (R1 * AL + R2) /NR1 → R3 * AL + R4
rule340 $ 60__69 : (R1 * AL) /NR1 → R2 * AL
rule200 $ 49__57 : (R1 * AL + R2) * R3 → NR1 * AL + R4
rule210 $ 47__36 : A * R1 + A * (−1) * R1 → 0
rule210 $ 66__37 : AL * R1 + AL * R2 → NR1 * AL
rule010 $ 30__34 : NR1 * AL + R1 = R2 * AL + R3 → 1 * AL = R4 (DS)
rule010 $ 41__35 : R1 * AL = R2 * AL + R3 → 1 * AL = R4 (DS)
rule010 $ 10__23 : NR1 * AL + R1 = R2 → 1 * AL = R3 (DS)
rule000 $ 3__22 : R1 = NR1 * AL → 1 * AL = R2 (DS)
rule340 $ 6__3 : (NA * AL) /NA → 1 * AL
rule030 $ 0__2 : NR1 * AL = R1 → 1 * AL = R2 (DS)

絶対オペレータ

rule310__67 : R1 * R2 → R3
rule230__56 : 0 * A → 0
rule300__33 : R1 + R2 → R3
rule270__14 : R1 − R1 → 0
rule320__21 : R1/NR1 → R2
rule280__13 : NA/NA → 1

DSBGによるDSマクロオペレータ

rule010 $ 41__35__ds : R1 * A = R2 * A + R3 → R4 * A = R5 (DS)
rule010 $ 30__34__ds : NR1 * AL + R1 = A + R2 → NR1 * AL = A + R3 (DS)
rule010 $ 10__23__ds : NRA + R1 = R2 → NRA = R3 (DS)
rule000 $ 3__22__ds : B = A → A = B (DS)

[2次方程式]

マクロオペレータ

rule000 $ 3__22__dsbg__447 : R1 * AL = NR1 * AL * AL → 1 * AL = R2 (DS)
rule000 $ 3__22__dsbg__441 : R1 = NR1 * AL * AL → AL = R2 (DS)
rule000 $ 3__22__dsbg__432 : R1 * AL * AL + R2 = R3 * AL * AL → AL = R4 (DS)
rule000 $ 3__22__dsbg__424 : R1 * AL * AL + NR1 * AL = R2 * AL * AL
→ 1 * AL = R3 (DS)
rule000 $ 3__22__dsbg__416 : R1 * AL * AL = R2 * AL + R3 → 1 * AL = R4 (DS)
rule000 $ 3__22__dsbg__400 : R1 = NR1 * AL * AL + R2 → AL = R3 (DS)
rule000 $ 3__22__dsbg__387 : R1 * AL + R2 = NR1 * AL * AL + R3 → 1 * AL = R4 (DS)
rule000 $ 3__22__dsbg__370 : R1 * AL * AL + R2 * AL = R3 * AL * AL + R4
→ 1 * AL = R5 (DS)
rule000 $ 3__22__dsbg__362 : R1 * AL * AL + R2 * AL + R3 = R4 * AL * AL + R5
→ 1 * AL = R6 (DS)
rule000 $ 3__22__dsbg__345 : R1 = NR1 * AL * AL + R2 * AL → 1 * AL = R3 (DS)
rule000 $ 3__22__dsbg__332 : R1 * AL + R2 = NR1 * AL * AL + R3 * AL → 1 * AL = R4 (DS)
rule010 $ 319__101__dsbg__325 : R1 * AL * AL + R2 = R3 * AL * AL + R4 * AL
→ 1 * AL = R5 (DS)
rule000 $ 3__22__dsbg__315 : R1 * AL * AL + R2 * AL + R3 = R4 * AL * AL + R5 * AL
→ 1 * AL = R6 (DS)
rule000 $ 3__22__dsbg__298 : R1 = NR1 * AL * AL + R2 * AL + R3 → 1 * AL = R4 (DS)
rule000 $ 3__22__dsbg__282 : R1 * AL + R2 = NR1 * AL * AL + R3 * AL + R4
→ 1 * AL = R5 (DS)
rule010 $ 10__23__dsbg__84 : (NR1 * AL + R1) * (NR1 * AL + R1) + R2 = R3
→ 1 * AL = R4 (DS)
rule010 $ 10__23__dsbg__77 : NR1 * AL * AL + R1 = R2 → AL = R3 (DS)

rule010 $ 456__113 : R1 * AL = R2 * AL → 1 * AL = R3 (DS)
rule200 $ 452__112 : (AL * AL + R1) * R2 → R2 * AL * AL + R3
rule351 $ 468__110 : (R1 * AL * AL + R2 * AL + R3)/NR1 → R4 * AL*AL + R5*AL + R6
rule342 $ 499__111 : (R1 * AL * AL) /NR1 → R2 * AL * AL
rule010 $ 436__109 : R1 * AL * AL = R2 * AL * AL → AL = R3 (DS)
rule010 $ 410__108 : NR1 * AL * AL + R1 = R2 * AL → 1 * AL = R3 (DS)
rule000 $ 405__107 : R1 * AL = NR1 * AL * AL + R2 → 1 * AL = R3 (DS)
rule010 $ 394__106 : R1 * AL * AL = R2 * AL * AL + R3 → AL = R4 (DS)
rule010 $ 379__105 : R1 * AL * AL + R2 = R3 * AL * AL + R4 → AL = R5 (DS)
rule010 $ 355__104 : NR1 * AL * AL + R1 * AL = R2 * AL → 1 * AL = R3 (DS)
rule000 $ 350__103 : R1 * AL = NR1 * AL * AL + R2 * AL → 1 * AL = R3 (DS)
rule010 $ 339__102 : R1 * AL * AL = R2 * AL * AL + NR1 * AL → 1 * AL = R3 (DS)
rule010 $ 319__101 : R1*AL* AL + R2 * AL = R3 * AL * AL + R4 * AL → 1 * AL = R5 (DS)
rule010 $ 308__100 : NR1 * AL * AL + R1 * AL + R2 = R3 * AL → 1 * AL = R4 (DS)
rule000 $ 303__99 : R1 * AL = NR1 * AL * AL + R2 * AL + R3 → 1 * AL = R4 (DS)
rule010 $ 292__98 : R1 * AL * AL + R2 * AL + R3 = R4 * AL * AL → 1 * AL = R5 (DS)
rule000 $ 287__97 : R1 * AL * AL = R2 * AL * AL + R3 * AL + R4 → 1 * AL = R5 (DS)

rule010 $ 264__95 : R1 * AL * AL + R2 = R3 * AL * AL + R4 * AL + R5 → 1 * AL = R6 (DS)
rule010 $ 276__96 : NR1 * AL * AL + R1 = R2 * AL + R3 → 1 * AL = R4 (DS)
rule010 $ 246__93 : R1*AL*AL + R2*AL = R3*AL*AL + R4* AL + R5 → 1 * AL = R6 (DS)
rule010 $ 255__94 : NR1 * AL * AL + R1 * AL = R2 * AL + R3 → 1 * AL = R4 (DS)
rule211__222 $ 234__91 : R1 * AL * AL + R2 * AL * AL → NR1 * AL * AL
rule010 $ 228__90 : R1*AL*AL + R2*AL + R3 = R4*AL*AL + R5* AL + R6
→ 1 * AL = R7 (DS)
rule010 $ 242__92 : NR1 * AL * AL + R1 * AL + R2 = R3 * AL + R4 → 1 * AL = R5 (DS)
rule352 $ 224__89 : (R1 * AL * AL + R2 * AL + R3)* R4 → NR1 * AL * AL + R5 * AL + R6
rule211 $ 222__88 : R1 * A * A − R1 * A * A → 0
rule010 $ 218__87 : R1 * AL + R2 = R3 * AL * AL → 1 * AL = R4 (DS)
rule010 $ 218__86 : AL * AL = R1 * AL + R2 → 1 * AL = R3 (DS)
rule010 $ 214__85 : AL * AL + R1 = R2 * AL → 1 * AL = R3 (DS)
rule010 $ 210__84 : AL * AL = R1 * AL → 1 * AL = R2 (DS)
rule010 $ 206__83 : NR1 * AL * AL = R1 * AL → 1 * AL = R2 (DS)
rule350 $ 202__82 : (NR1 * AL * AL + R1 * AL) /NR1 → AL * AL + R2 * AL
rule030 $ 195__81 : NR1 * AL * AL + R1 * AL = R2 → 1 * AL = R3 (DS)
rule200__193 $ 195__80 : (1 * AL + R1) * AL → AL * AL + R1 * AL
rule200 $ 166__79 : (R1 * AL + R2)*(R3 * AL + R4) → NR1 * AL * AL + NR2 * AL + R5
rule351 $ 165__78 : (NR1 * AL * AL + R1 * AL + R2) /NR1 → AL * AL + R3 * AL + R4
rule030 $ 151__77 : NR1 * AL * AL + R1 * AL + R2 = R3 → 1 * AL = R4 (DS)
rule010 $ 150__76 : AL * AL + R1 * AL + R2 = R3 → 1 * AL = R4 (DS)
rule430 $ 122__75 : R1 * AL → 2 * R2 * AL
rule010 $ 88__74 : AL * AL + R1 * AL = R2 → 1 * AL = R3 (DS)
rule440 $ 78__73 : (NR1 * AL + R1) * (NR1 * AL + R1) = A → 1 * AL = R2 (DS)
rule342 $ 63__72 : (NA * AL * AL) /NA → AL * AL
rule030 $ 55__71 : NR1 * AL * AL = R1 → AL = R2 (DS)
rule010 $ 59__70 : AL * AL + R1 = R2 → AL = R3 (DS)

DSBGによるDSマクロオペレータ

rule010 $ 456__113__ds : R1 * AL = R2 * AL → NR1 * AL = 0 (DS)
rule010 $ 436__109__ds : R1 * AL * AL = R2 * AL * AL → NR1 * AL * AL = 0 (DS)
rule010 $ 410__108__ds : A * C * B + E = R1 * D → A * B * C − R1 * D + E = 0 (DS)
rule010 $ 394__106__ds : R1 * AL * AL = R2 * AL * AL + R3 → NR1 * AL * AL = R4 (DS)

rule010 $ 379__105__ds : R1 * AL * AL + A = R2 * AL * AL + R3
→ NR1 * AL * AL + A = R4 (DS)
rule010 $ 355__104__ds : A * B * C + R1 * AL = R2 * AL → A * B * C + NR1 * AL = 0 (DS)
rule010 $ 339__102__ds : R1 * AL * AL = R2 * AL * AL + NR2 * AL
→ NR1 * AL * AL = NR2 * AL (DS)
rule010 $ 319__101__ds : R1 * AL * AL + A = R2 * AL * AL + B
→ NR1 * AL * AL + A = B + 0 (DS)

rule010 $ 308__100__ds : A * B * C + R1 * AL + D = R2 * AL
→ A * B * C + NR1 * AL + D = 0 (DS)
rule010 $ 292__98__ds : R1 * AL * AL + A + B = R2 * AL * AL
→ NR1 * AL * AL + A + B = 0 (DS)
rule010 $ 276__96__ds : A * B * C + E = R1 * D + R2 → A * B * C − R1 * D + E = R4 (DS)
rule010 $ 264__95__ds : R1 * AL * AL + A = R2 * AL * AL + B + R3
→ NR1 * AL * AL + A = B + R4 (DS)
rule010 $ 255__94__ds : A * C * B + R1 * AL = R2 * AL + R3
→ A * B * C + NR1 * AL = R4 (DS)
rule010 $ 246__93__ds : R1 * AL * AL + A = R2 * AL * AL + B + R3
→ NR1 * AL * AL + A = B + R4 (DS)
rule010 $ 242__92__ds : A * C * B + R1 * AL + D = R2 * AL + R3
→ A * B * C + NR1 * AL + D = R4 (DS)
rule010 $ 228__90__ds : R1 * A * A + B + C = R2 * A * A + D + R3
→ R4 * A * A + B + C = D + R5 (DS)
rule010 $ 218__87__ds : B + C = R1 * A * A → (−1) * R1 * A * A + B + C = 0 (DS)
rule010 $ 218__86__ds : A * B = R1 * C + R2 → A * B − R1 * C = R4 (DS)
rule010 $ 214__85__ds : B * A + D = R1 * C → A * B − R1 * C + D = 0 (DS)
rule010 $ 210__84__ds : B * A = R1 * C → A * B − R1 * C = 0 (DS)
rule010 $ 206__83__ds : A * C * B = R1 * D → A * B * C − R1 * D = 0 (DS)
rule030 $ 195__81__ds : NR1 * AL * AL + R1 * AL = R2 → AL * AL + R3 * AL = R4 (DS)
rule030 $ 151__77__ds : NR1 * AL * AL + R1 * AL + R2 = R3
→ AL * AL + R4 * AL + R5 = R6 (DS)
rule010 $ 150__76__ds : A * B + C + R1 = R2 → A * B + C = R3 (DS)
rule010 $ 88__74__ds : A * A + R1 * A = R2 → (1 * A + R3) * (1 * A + R3) = R4 (DS)
rule440 $ 78__73__ds : B * B = A → B = C (DS)

[分数方程式]

マクロオペレータ

rule020 $ 680__124__dsbg__718 : (NR1 * AL * AL) ／AL = R1 → 1 * AL = R2 (DS)
rule020 $ 680__124__dsbg__712 : (NR1 * AL * AL + R1 * AL) ／AL = R2 → 1 * AL = R3 (DS)

rule020 $ 680__124__dsbg__705 : (NR1 * AL * AL + R1 * AL + R2) ／AL = R3
→ 1 * AL = R4 (DS)
rule020 $ 680__124__dsbg__699 : (NR1 * AL * AL + R1) ／AL = R2 → 1 * AL = R3 (DS)
rule020 $ 680__124__dsbg__693 : (R1 * AL + R2) ／AL = R3 → 1 * AL = R4 (DS)
rule020 $ 680__124__dsbg__687 : (R1 * AL) ／AL = R2 → 1 * AL = R3 (DS)
rule020 $ 643__122__dsbg__678 : (R1 * AL * AL) ／ (R2 * AL + R3) = R4 * AL + R5
→ 1 * AL = R6 (DS)
rule020 $ 643__122__dsbg__673 : (R1 *AL*AL+ R2 * AL)／(R3 * AL + R4)= R5 * AL + R6
→ 1 * AL = R7 (DS)

rule020 $ 643__122__dsbg__666 : (R1*AL*AL + R2*AL + R3)／(R4*AL + R5)= R6*AL + R7
→ 1 * AL = R8 (DS)

rule020 $ 643__122__dsbg__660 : (R1 * AL * AL + R2)／(R3 * AL + R4)= R5*AL + R6
→ 1 * AL = R7 (DS)

rule020 $ 643__122__dsbg__653 : (R1 * AL + R2) ／ (R3 * AL + R4) = R5 * AL + R6
→ 1 * AL = R7 (DS)

rule020 $ 643__122__dsbg__648 : (R1 * AL) ／ (R2 * AL + R3) = R4 * AL + R5
→ 1 * AL = R6 (DS)

rule020 $ 572__119__dsbg__641 : (R1 * AL * AL) ／ (R2 * AL * AL + R3 * AL) = R4
→ 1 * AL = R5 (DS)

rule020 $ 572__119__dsbg__635 : (R1 * AL * AL + R2 * AL)／(R3*AL*AL + R4*AL)= R5
→ 1 * AL = R6 (DS)

rule020 $ 572__119__dsbg__629 : (R1*AL*AL+ R2*AL + R3)／(R4*AL*AL + R5*AL)= R6
→ 1 * AL = R7 (DS)

rule020 $ 572__119__dsbg__624 : (R1*AL*AL + R2)／(R3*AL*AL + R4*AL) = R5
→ 1 * AL = R6 (DS)

rule020 $ 572__119__dsbg__618 : (R1 * AL + R2) ／ (R3 * AL * AL + R4 * AL) = R5
→ 1 * AL = R6 (DS)

rule020 $ 572__119__dsbg__613 : (R1 * AL)／(R2 * AL * AL + R3 * AL) = R4
→ 1 * AL = R5 (DS)

rule020 $ 498__116__dsbg__570 : (R1 * AL * AL)／(R2 * AL * AL + R3) = R4 → AL = R5 (DS)

rule020 $ 498__116__dsbg__564 : (R1 * AL * AL + R2 * AL)／(R3 * AL * AL + R4)= R5
→ 1 * AL = R6 (DS)

rule020 $ 498__116__dsbg__557 : (R1 * AL*AL + R2*AL + R3)／(R4 * AL * AL + R5)= R6
→ 1 * AL = R7 (DS)

rule020 $ 498__116__dsbg__552 : (R1 * AL * AL + R2) ／ (R3 * AL * AL + R4) = R5
→ AL = R6 (DS)

rule020 $ 498__116__dsbg__545 : (R1 * AL + R2) ／ (R3 * AL * AL + R4) = R5
→ 1 * AL = R6 (DS)

rule020 $ 498__116__dsbg__539 : (R1 * AL) ／ (R2 * AL * AL + R3) = R4
→ 1 * AL = R5 (DS)

rule020 $ 449__114__dsbg__496 : (R1 * AL * AL) ／ (R2 * AL + R3) = R4
→ 1 * AL = R5 (DS)

rule020 $ 449__114__dsbg__491 : (NR1 * AL * AL + R1 * AL)／(R2 * AL + R3) = R4
→ 1 * AL = R5 (DS)

rule020 $ 449__114__dsbg__484 : (NR1 * AL * AL + R1 * AL + R2)／(R3 * AL + R4) = R5
→ 1 * AL = R6 (DS)

rule020 $ 449__114__dsbg__478 : (NR1 * AL * AL + R1)／(R2 * AL + R3) = R4
→ 1 * AL = R5 (DS)

rule020 $ 449__114__dsbg__472 : (R1 * AL + R2)／(R3 * AL + R4) = R5 → 1 * AL = R6 (DS)

rule020 $ 449__114__dsbg__465 : (R1 * AL)／(R2 * AL + R3) = R4 → 1 * AL = R5 (DS)

rule020 $ 680__124 : R1/AL = NR1 → 1 * AL = R2 (DS)
rule345 $ 684__125 : NA * (R1/NA) → R2
rule020 $ 643__122 : R1/ (R2 * AL + R3) = R4 * AL + R5 → 1 * AL = R6 (DS)
rule345 $ 650__123 : (A * B + C) * (R1/ (A * B + C)) → R2
rule020 $ 572__119 : R1/ (R2 * AL * AL + R3 * AL) = R4 → 1 * AL = R5 (DS)
rule345 $ 577__120 : (A * C * C + B * C) * (R1/ (A * C * C + B * C)) → R2
rule200 $ 608__121 : (R1 * AL * AL + R2 * AL) * R3 → NR1 * AL * AL + R4 * AL
rule020 $ 498__116 : R1/ (R2 * AL * AL + R3) = R4 → AL = R5 (DS)
rule345 $ 502__117 : (A * B * B + C) * (R1/ (A * B * B + C)) → R2
rule200 $ 534__118 : (R1 * AL * AL + R2) * R3 → NR1 * AL * AL + R4
rule020 $ 449__114 : R1/ (R2 * AL + R3) = R4 → 1 * AL = R5 (DS)
rule345 $ 458__115 : (A * B + C) * (R1/ (A * B + C)) → R2

DSBGによるマクロオペレータ

rule020 $ 680__124__ds : A/NA = B → A = B * NA (DS)
rule020 $ 643__122__ds : A/ (R1 * AL + R2) = R3 * AL + R4
→ A = NR1 * AL * AL + NR2 * AL + R5 (DS)
rule020 $ 572__119__ds : A/ (R1 * B * C + R2 * D) = R3 → A = R4 * B * C + R5 * D (DS)
rule020 $ 498__116__ds : A/ (R1 * B * C + R2) = R3 → A = R4 * B * C + R5 (DS)
rule020 $ 449__114__ds : A/ (R1 * AL + R2) = R3 → A = NR1 * AL + R4 (DS)

[対数方程式]

マクロオペレータ

rule000 $ 3__22__dsbg__788 : R1 = log (R2,(NR1 * AL * AL + R3 * AL + R4)) + R5
→ 1 * AL = R6 (DS)
rule000 $ 3__22__dsbg__782 : R1 = log (R2,(NR1 * AL * AL + R3 * AL)) + R4
→ 1 * AL = R5 (DS)
rule000 $ 3__22__dsbg__776 : R1 = log (R2,(NR1 * AL * AL + R3)) + R4 → AL = R5 (DS)
rule000 $ 3__22__dsbg__770 : R1 = log (R2,NR1 * AL * AL) + R3 → AL = R4 (DS)
rule010 $ 10__23__dsbg__764 : log (R1,(NR1 * AL * AL + R2 * AL + R3)) + R4 = R5
→ 1 * AL = R6 (DS)
rule010 $ 10__23__dsbg__758 : log (R1,(NR1 * AL * AL + R2 * AL)) + R3 = R4
→ 1 * AL = R5 (DS)
rule010 $ 10__23__dsbg__752 : log (R1,(NR1 * AL * AL + R2)) + R3 = R4 → AL = R5 (DS)
rule010 $ 10__23__dsbg__746 : log (R1,NR1 * AL * AL) + R2 = R3 → AL = R4 (DS)
rule110 $ 502__459__dsbg__740 : log (A,(R1 * AL * AL + R2 * AL + R3))
= log (A,(R4 * AL * AL + R5 * AL + R6)) → 1 * AL = R7 (DS)
rule110 $ 502__459__dsbg__734 : log (A,(R1 * AL * AL + R2))
= log (A,(R3 * AL * AL + R4 * AL + R5)) → 1 * AL = R6 (DS)
rule110 $ 502__459__dsbg__727 : log (A,(R1 * AL * AL + R2 * AL))

= log (A,(R3 * AL * AL + R4 * AL + R5)) → 1 * AL = R6 (DS)

rule110 $ 502__459__dsbg__720 : log (A,R1 * AL * AL)
= log (A,(R2 * AL * AL + R3 * AL + R4)) → 1 * AL = R5 (DS)

rule110 $ 502__459__dsbg__715 : log (A,(R1 * AL * AL + R2 * AL + R3))
= log (A,(R4 * AL * AL + R5 * AL)) → 1 * AL = R6 (DS)

rule110 $ 502__459__dsbg__710 : log (A,(R1 * AL * AL + R2))
= log (A,(R3 * AL * AL + R4 * AL)) → 1 * AL = R5 (DS)

rule110 $ 502__459__dsbg__704 : log (A,(R1 * AL * AL + R2 * AL))
= log (A,(R3 * AL * AL + R4 * AL)) → 1 * AL = R5 (DS)

rule110 $ 502__459__dsbg__698 : log (A,R1 * AL * AL)
= log (A,(R2 * AL * AL + NR1 * AL)) → 1 * AL = R3 (DS)

rule110 $ 502__459__dsbg__692 : log (A,(R1 * AL * AL + R2 * AL + R3))
= log (A,(R4 * AL * AL + R5)) → 1 * AL = R6 (DS)

rule110 $ 502__459__dsbg__687 : log (A,(R1 * AL * AL + R2)) = log (A,(R3 * AL * AL + R4))
→ AL = R5 (DS)

rule110 $ 502__459__dsbg__680 : log (A,(R1 * AL * AL + R2 * AL))
= log (A,(R3 * AL * AL + R4)) → 1 * AL = R5 (DS)

rule110 $ 502__459__dsbg__673 : log (A,R1 * AL * AL) = log (A,(R2 * AL * AL + R3))
→ AL = R4 (DS)

rule110 $ 502__459__dsbg__667 : log(A,(R1*AL*AL+R2*AL+R3))= log(A,R4*AL*AL)
→ 1 * AL = R5 (DS)

rule110 $ 502__459__dsbg__661 : log (A,(R1 * AL * AL + R2)) = log (A,R3 * AL * AL)
→ AL = R4 (DS)

rule110 $ 502__459__dsbg__655 : log (A,(R1 * AL * AL + NR1 * AL)) = log (A,R2 * AL * AL)
→ 1 * AL = R3 (DS)

rule110 $ 502__459__dsbg__649 : log (A,R1 * AL * AL) = log (A,R2 * AL * AL)
→ AL = R3 (DS)

rule110 $ 502__459__dsbg__643 : log (A,(NR1 * AL * AL + R1 * AL + R2))
= log (A,(R3 * AL + R4)) → 1 * AL = R5 (DS)

rule110 $ 502__459__dsbg__637 : log (A,(NR1 * AL * AL + R1)) = log (A,(R2 * AL + R3))
→ 1 * AL = R4 (DS)

rule110 $ 502__459__dsbg__631 : log(A,(NR1*AL*AL + R1*AL))= log(A,(R2*AL + R3))
→ 1 * AL = R4 (DS)

rule110 $ 502__459__dsbg__624 : log (A,R1 * AL * AL) = log (A,(R2 * AL + R3))
→ 1 * AL = R4 (DS)

rule110 $ 502__459__dsbg__619 : log(A,(NR1*AL*AL + R1*AL + R2))= log(A,R3 * AL)
→ 1 * AL = R4 (DS)

rule110 $ 502__459__dsbg__613 : log (A,(NR1 * AL * AL + R1)) = log (A,R2 * AL)
→ 1 * AL = R3 (DS)

rule110 $ 502__459__dsbg__607 : log (A,(NR1 * AL * AL + R1 * AL)) = log (A,R2 * AL)
→ 1 * AL = R3 (DS)

rule110 $ 502__459__dsbg__600 : log (A,NR1 * AL * AL) = log (A,R1 * AL)

→ 1 * AL = R2 (DS)

rule110 $ 502__459__dsbg__594 : log (A,R1 * AL) = log (A,(R2 * AL + R3))
→ 1 * AL = R4 (DS)

rule110 $ 502__459__dsbg__587 : log (A,(R1 * AL + R2)) = log (A,R3 * AL)
→ 1 * AL = R4 (DS)

rule110 $ 502__459__dsbg__581 : log (A,R1 * AL) = log (A,R2 * AL) → 1 * AL = R3 (DS)

rule000 $ 3__22__dsbg__574 : R1 = log (R2,(NR1 * AL * AL + R3 * AL + R4))
→ 1 * AL = R5 (DS)

rule000 $ 3__22__dsbg__568 : R1 = log (R2,(NR1 * AL * AL + R3)) → AL = R4 (DS)

rule000 $ 3__22__dsbg__562 : R1 = log (R2,NR1 * AL * AL) → AL = R3 (DS)

rule000 $ 3__22__dsbg__556 : R1 = log (R2,(NR1 * AL * AL + R3 * AL)) → 1 * AL = R4 (DS)

rule000 $ 3__22__dsbg__549 : R1 = log (R2,(NR1 * AL + R3)) + R4 → 1 * AL = R5 (DS)

rule000 $ 3__22__dsbg__543 : R1 = log (R2,NR1 * AL) + R3 → 1 * AL = R4 (DS)

rule110 $ 502__459__dsbg__537 : log (A,R1) = log (A,NR1 * AL) → 1 * AL = R2 (DS)

rule010 $ 10__23__dsbg__530 : log (R1,(NR1 * AL + R2)) + R3 = R4 → 1 * AL = R5 (DS)

rule010 $ 10__23__dsbg__524 : log (R1,NR1 * AL) + R2 = R3 → 1 * AL = R4 (DS)

rule110 $ 502__459__dsbg__518 : log (A,R1) = log (A,(NR1 * AL + R2)) → 1 * AL = R3 (DS)

rule110 $ 502__459__dsbg__513 : log (A,(NR1 * AL + R1)) = log (A,R2) → 1 * AL = R3 (DS)

rule110 $ 502__459__dsbg__507 : log (A,NR1 * AL) = log (A,R1) → 1 * AL = R2 (DS)

rule000 $ 3__22__dsbg__499 : R1 = log (R2,(NR1 * AL + R3)) → 1 * AL = R4 (DS)

rule000 $ 3__22__dsbg__493 : R1 = log (R2,NR1 * AL) → 1 * AL = R3 (DS)

rule111 $ 449__114__dsbg__487 : log(R1,(NR1*AL*AL + R2*AL + R3)) = R4
→ 1*AL = R5 (DS)

rule111 $ 449__114__dsbg__481 : log (R1,(NR1 * AL * AL + R2 * AL)) = R3
→ 1 * AL = R4 (DS)

rule111 $ 449__114__dsbg__475 : log (R1,(NR1 * AL * AL + R2)) = R3 → AL = R4 (DS)

rule111 $ 449__114__dsbg__469 : log (R1,NR1 * AL * AL) = R2 → AL = R3 (DS)

rule111 $ 449__114__dsbg__464 : log (R1,(NR1 * AL + R2)) = R3 → 1 * AL = R4 (DS)

rule103 $ 532__460 : log (R4,R1 * AL) + log (R4,(R2 * AL + R3))
→ log (R4,(NR1 * AL * AL + R5 * AL))

rule110 $ 502__459 : log (A,(R1 * AL + R2)) = log (A,(NR1 * AL + R3)) → 1 * AL = R4 (DS)

rule111 $ 449__114 : log (R1,NR1 * AL) = R2 → 1 * AL = R3 (DS)

絶対オペレータ

rule127__458 : exp (R1,R2) → R3

DSBGによるDSマクロオペレータ

rule110 $ 502__459__ds : log (A,B) = log (A,C) → B = C (DS)

rule111 $ 449__114__ds : log (R1,A) = R2 → A = R3 (DS)

[指数方程式]

マクロオペレータ

rule000 $ 3__22__dsbg__758 : R1 = exp (R2,(NR1 * AL * AL + R3 * AL + R4)) + R5
→ 1 * AL = R6 (DS)

rule000 $ 3__22__dsbg__754 : R1 = exp (R2,(NR1 * AL * AL + R3 * AL)) + R4
→ 1 * AL = R5 (DS)

rule000 $ 3__22__dsbg__750 : R1 = exp (R2,(NR1 * AL * AL + R3)) + R4 → AL = R5 (DS)

rule000 $ 3__22__dsbg__746 : R1 = exp (R2,NR1 * AL * AL) + R3 → AL = R4 (DS)

rule010 $ 10__23__dsbg__742 : exp (R1,(NR1 * AL * AL + R2 * AL + R3)) + R4 = R5
→ 1 * AL = R6 (DS)

rule010 $ 10__23__dsbg__738 : exp (R1,(NR1 * AL * AL + R2 * AL)) + R3 = R4
→ 1 * AL = R5 (DS)

rule010 $ 10__23__dsbg__734 : exp (R1,(NR1 * AL * AL + R2)) + R3 = R4 → AL = R5 (DS)

rule010 $ 10__23__dsbg__730 : exp (R1,NR1 * AL * AL) + R2 = R3 → AL = R4 (DS)

rule126 $ 499__115__dsbg__726 : exp (A,(R1 * AL * AL + R2 * AL + R3))
= exp (A,(R4 * AL * AL + R5 * AL + R6)) → 1 * AL = R7 (DS)

rule126 $ 499__115__dsbg__722 : exp (A,(R1 * AL * AL + R2))
= exp (A,(R3 * AL * AL + R4 * AL + R5)) → 1 * AL = R6 (DS)

rule126 $ 499__115__dsbg__718 : exp (A,(R1 * AL * AL + R2 * AL))
= exp (A,(R3 * AL * AL + R4 * AL + R5)) → 1 * AL = R6 (DS)

rule126 $ 499__115__dsbg__714 : exp (A,R1 * AL * AL)
= exp (A,(R2 * AL * AL + R3 * AL + R4)) → 1 * AL = R5 (DS)

rule126 $ 499__115__dsbg__710 : exp (A,(R1 * AL * AL + R2 * AL + R3))
= exp (A,(R4 * AL * AL + R5 * AL)) → 1 * AL = R6 (DS)

rule126 $ 499__115__dsbg__706 : exp (A,(R1 * AL * AL + R2))
= exp (A,(R3 * AL * AL + R4 * AL)) → 1 * AL = R5 (DS)

rule126 $ 499__115__dsbg__700 : exp (A,(R1 * AL * AL + R2 * AL))
= exp (A,(R3 * AL * AL + R4 * AL)) → 1 * AL = R5 (DS)

rule126 $ 499__115__dsbg__694 : exp (A,R1 * AL * AL) = exp(A,(R2 * AL * AL + NR1 * AL))
→ 1 * AL = R3 (DS)

rule126 $ 499__115__dsbg__688 : exp (A,(R1 * AL * AL + R2 * AL + R3))
= exp (A,(R4 * AL * AL + R5)) → 1 * AL = R6 (DS)

rule126 $ 499__115__dsbg__683 : exp (A,(R1 * AL * AL + R2)) = exp (A,(R3 * AL * AL + R4))
→ AL = R5 (DS)

rule126 $ 499__115__dsbg__676 : exp (A,(R1 * AL * AL + R2 * AL))
= exp (A,(R3 * AL * AL + R4)) → 1 * AL = R5 (DS)

rule126 $ 499__115__dsbg__669 : exp (A,R1 * AL * AL) = exp (A,(R2 * AL * AL + R3))
→ AL = R4 (DS)

rule126 $ 499__115__dsbg__663 : exp(A,(R1*AL*AL + R2*AL + R3))= exp (A,R4 * AL * AL)
→ 1 * AL = R5 (DS)

rule126 $ 499__115__dsbg__657 : exp (A,(R1 * AL * AL + R2)) = exp (A,R3 * AL * AL)

→ AL = R4 (DS)

rule126 $ 499__115__dsbg__651 : exp(A,(R1 * AL * AL + NR1 * AL)) = exp(A,R2 * AL * AL)
→ 1 * AL = R3 (DS)

rule126 $ 499__115__dsbg__645 : exp (A,R1 * AL * AL) = exp (A,R2 * AL * AL)
→ AL = R3 (DS)

rule126 $ 499__115__dsbg__639 : exp (A,(NR1 * AL * AL + R1 * AL + R2))
= exp (A,(R3 * AL + R4)) → 1 * AL = R5 (DS)

rule126 $ 499__115__dsbg__633 : exp (A,(NR1 * AL * AL + R1)) = exp (A,(R2 * AL + R3))
→ 1 * AL = R4 (DS)

rule126 $ 499__115__dsbg__627 : exp(A,(NR1*AL*AL + R1 * AL)) = exp(A,(R2*AL+ R3))
→ 1 * AL = R4 (DS)

rule126 $ 499__115__dsbg__620 : exp (A,R1 * AL * AL) = exp (A,(R2 * AL + R3))
→ 1 * AL = R4 (DS)

rule126 $ 499__115__dsbg__615 : exp (A,(NR1 * AL * AL + R1 * AL + R2)) = exp (A,R3 * AL)
→ 1 * AL = R4 (DS)

rule126 $ 499__115__dsbg__609 : exp (A,(NR1 * AL * AL + R1)) = exp (A,R2 * AL)
→ 1 * AL = R3 (DS)

rule126 $ 499__115__dsbg__603 : exp (A,(NR1 * AL * AL + R1 * AL)) = exp (A,R2 * AL)
→ 1 * AL = R3 (DS)

rule126 $ 499__115__dsbg__596 : exp (A,NR1 * AL * AL) = exp (A,R1 * AL)
→ 1 * AL = R2 (DS)

rule126 $ 499__115__dsbg__590 : exp (A,R1 * AL) = exp (A,(R2 * AL + R3))
→ 1 * AL = R4 (DS)

rule126 $ 499__115__dsbg__583 : exp (A,(R1 * AL + R2)) = exp (A,R3 * AL)
→ 1 * AL = R4 (DS)

rule126 $ 499__115__dsbg__577 : exp (A,R1 * AL) = exp (A,R2 * AL) → 1 * AL = R3 (DS)

rule000 $ 3__22__dsbg__570 : R1 = exp (R2,(NR1 * AL * AL + R3 * AL + R4))
→ 1 * AL = R5 (DS)

rule000 $ 3__22__dsbg__564 : R1 = exp (R2,(NR1 * AL * AL + R3 * AL)) → 1 * AL = R4 (DS)

rule000 $ 3__22__dsbg__558 : R1 = exp (R2,(NR1 * AL * AL + R3)) → AL = R4 (DS)

rule000 $ 3__22__dsbg__552 : R1 = exp (R2,NR1 * AL * AL) → AL = R3 (DS)

rule000 $ 3__22__dsbg__546 : R1 = exp (R2,(NR1 * AL + R3)) + R4 → 1 * AL = R5 (DS)

rule000 $ 3__22__dsbg__540 : R1 = exp (R2,NR1 * AL) + R3 → 1 * AL = R4 (DS)

rule126 $ 499__115__dsbg__534 : exp (A,R1) = exp (A,NR1 * AL) → 1 * AL = R2 (DS)

rule010 $ 10__23__dsbg__527 : exp (R1,(NR1 * AL + R2)) + R3 = R4 → 1 * AL = R5 (DS)

rule010 $ 10__23__dsbg__521 : exp (R1,NR1 * AL) + R2 = R3 → 1 * AL = R4 (DS)

rule126 $ 499__115__dsbg__515 : exp (A,R1) = exp (A,(NR1 * AL + R2)) → 1 * AL = R3 (DS)

rule126 $ 499__115__dsbg__510 : exp (A,(NR1 * AL + R1)) = exp (A,R2) → 1 * AL = R3 (DS)

rule126 $ 499__115__dsbg__504 : exp (A,NR1 * AL) = exp (A,R1) → 1 * AL = R2 (DS)

rule000 $ 3__22__dsbg__497 : R1 = exp (R2,(NR1 * AL + R3)) → 1 * AL = R4 (DS)

rule000 $ 3__22__dsbg__491 : R1 = exp (R2,NR1 * AL) → 1 * AL = R3 (DS)

rule128 $ 449__114__dsbg__485 : exp (R1,(NR1 * AL * AL + R2 * AL + R3)) = R4

→ 1 * AL = R5 (DS)

rule128 $ 449__114__dsbg__479 : exp(R1,(NR1*AL*AL+R2*AL)) =R3 → 1*AL=R4 (DS)

rule128 $ 449__114__dsbg__473 : exp (R1,(NR1 * AL * AL + R2)) = R3 → AL = R4 (DS)

rule128 $ 449__114__dsbg__467 : exp (R1,NR1 * AL * AL) = R2 → AL = R3 (DS)

rule128 $ 449__114__dsbg__462 : exp (R1,(NR1 * AL + R2)) = R3 → 1 * AL = R4 (DS)

rule126 $ 499__115 : exp(A,(R1 * AL + R2)) = exp (A,(NR1 * AL + R3)) → 1 * AL = R4 (DS)

rule128 $ 449__114 : exp (R1,NR1 * AL) = R2 → 1 * AL = R3 (DS)

rule121 $ 529__116 : (exp (R3,R1*AL))*(exp(R3,(R2*AL+R4))) → exp(R3,(NR1 * AL + R4))

DSBGによるDSマクロオペレータ

rule126 $ 499__115__ds : exp (A,B) = exp (A,C) → B = C (DS)

rule128 $ 449__114__ds : exp (R1,A) = R2 → A = R3 (DS)

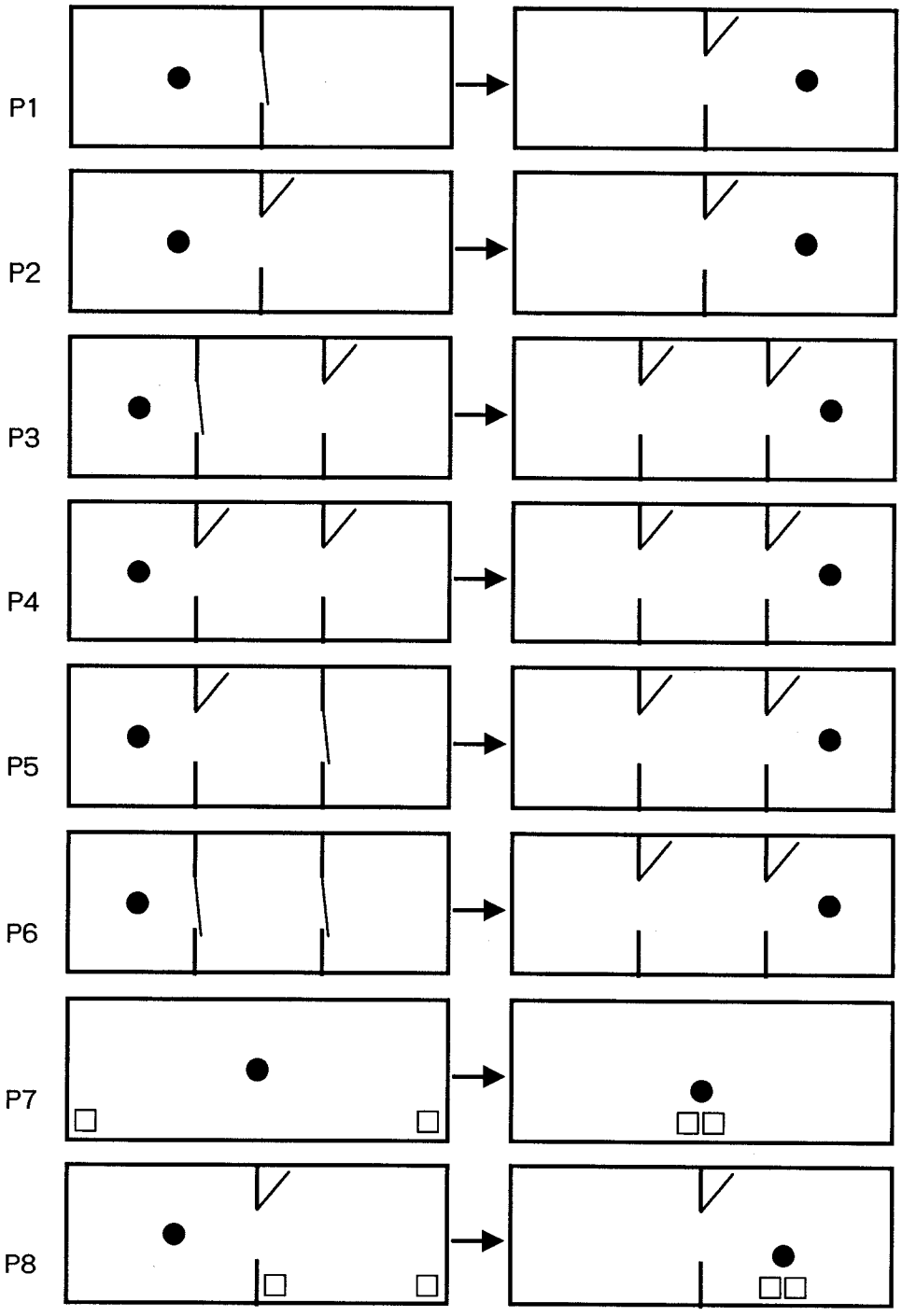
< Appendix6 >　PiL2の基本オペレータ

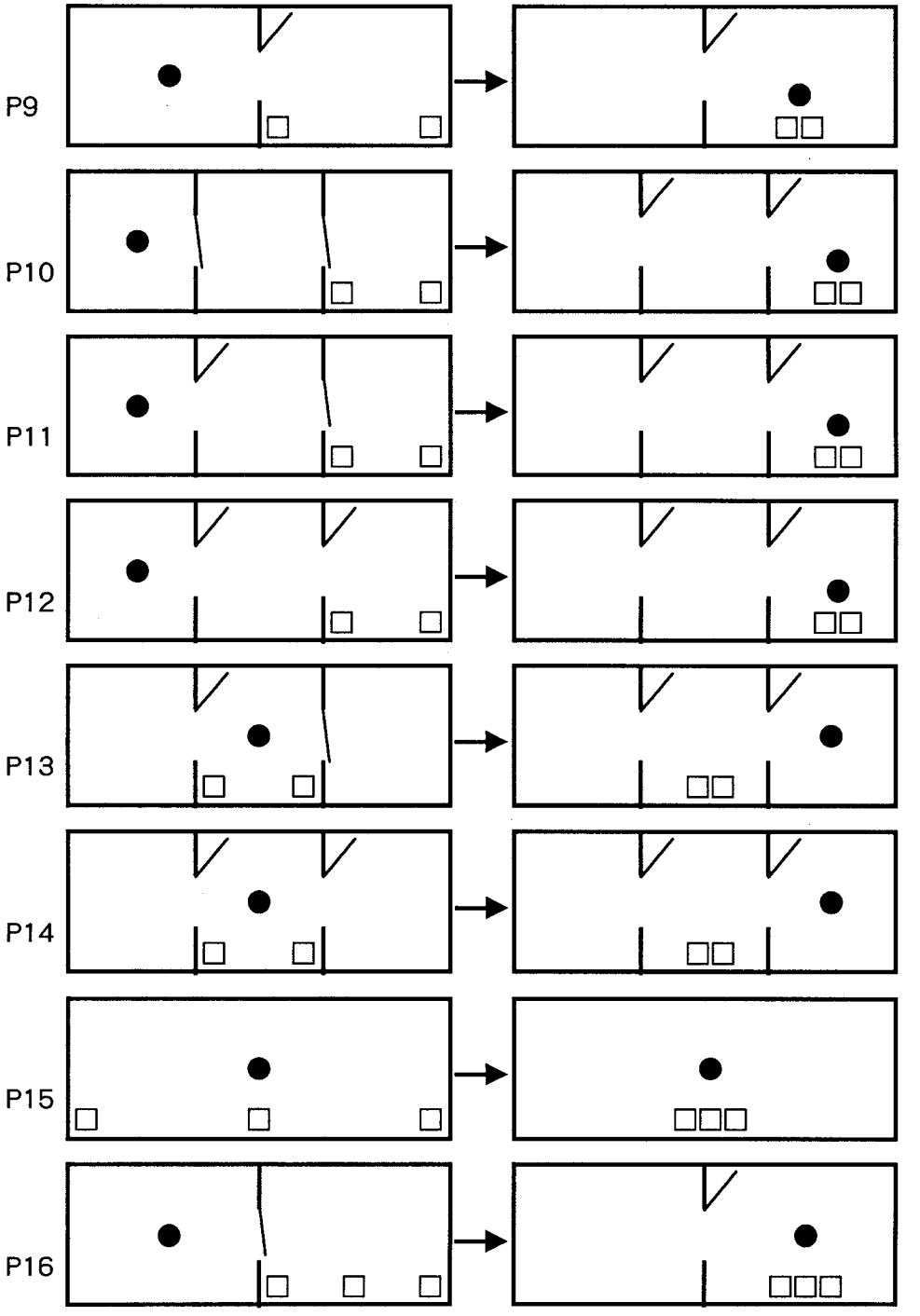
```
gotob (BX,RX)   {Go to object BX}
        [type (BX,object), inroom (BX,RX), inroom (robot,RX)],
        [nextto (robot,_), nextto (_,robot)],
        [nextto (robot,BX)],
        [],
        []

gotod (DX,RX)   {Go to door DX}
        [type (DX,door), inroom (robot,RX), connects (DX,RX,RY)],
        [nextto (robot,_), nextto (_,robot)],
        [nextto (robot,DX)],
        [],
        []

pushb (BX,BY,RX)   {Push BX to object BY}
        [type (BX,object), type (BY,object), pushable (BX),
         nextto (robot,BX), inroom (BX,RX), inroom (BY,RX)],
        [nextto (robot,_), nextto (_,robot), nextto (BX,_),
         nextto (_,BX)],
        [nextto (BX,BY), nextto (robot,BX)],
        [],
        [nextto (BX,BY)]

pushd (BX,DX,RX)   {Push BX to door DX}
        [pushable (BX), type (DX,door), nextto (robot,BX),
         inroom (BX,RX), connects (DX,RX,RY)],
        [nextto (robot,_), nextto (_,robot), nextto (BX,_),
         nextto (_,BX)],
        [nextto (BX,DX), nextto (robot,BX)],
        [],
```

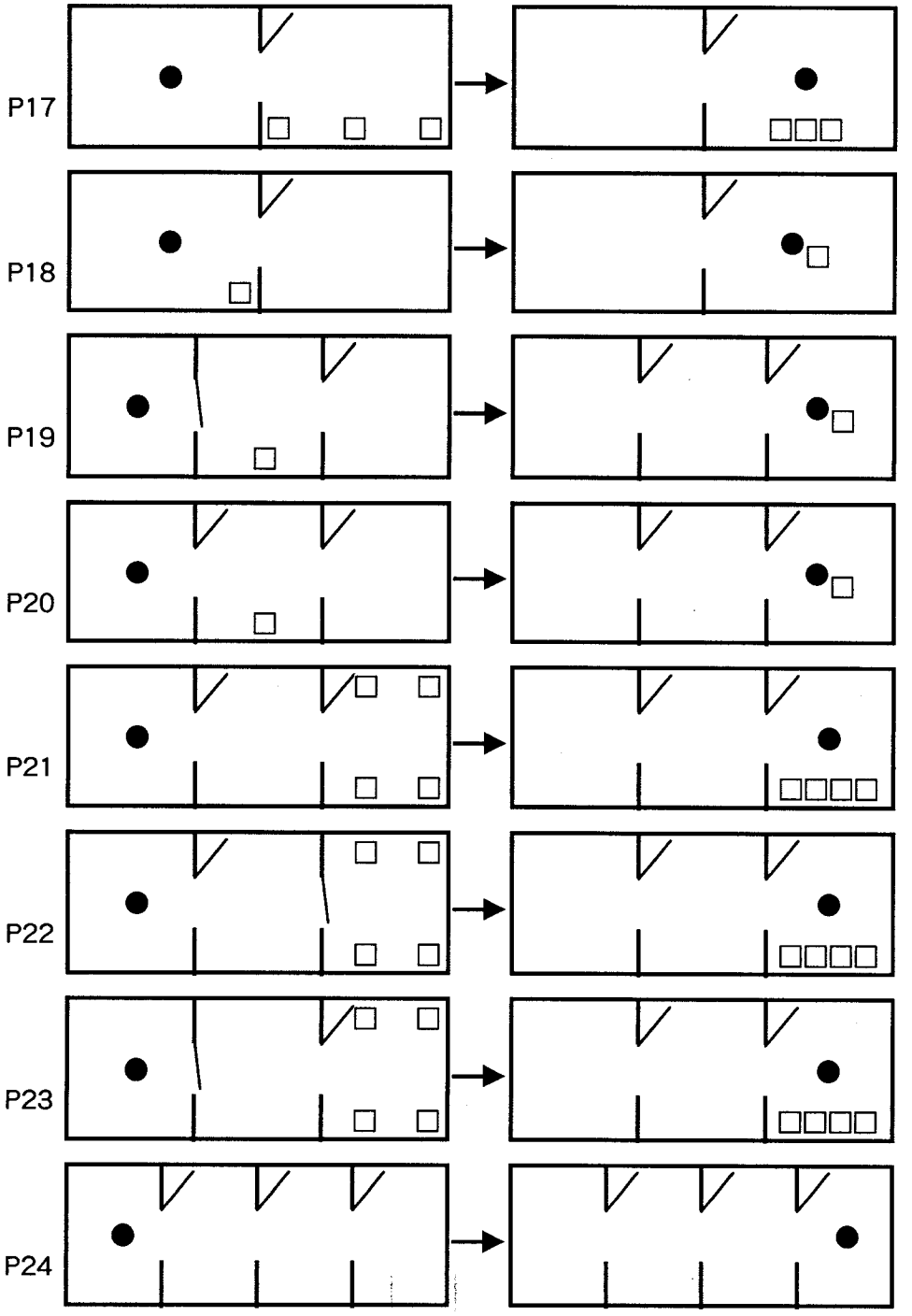
```
        [nextto (BX,DX)]

gothrudr (DX,RX,RY)   {GO through door DX into RX}
        [type (DX,door), status (DX,open), type (RX,room),
         nextto (robot,DX), inroom (robot,RY), connects (DX,RY,RX)],
        [inroom (robot,_), nextto (robot,_), nextto (_,robot)],
        [inroom (robot,RX)],
        [],
        []

pushthrudr (BX,DX,RX,RY)   {Push BX through door DX into room RX}
        [pushable (BX), type (DX,door), status (DX,open), type (RX,room),
         nextto (BX,DX), nextto (robot,BX), inroom (BX,RY),
         connects (DX,RY,RX)],
        [inroom (robot,_),inroom (BX,_),nextto (_,robot),
         nextto (robot,_),nextto (BX,_),nextto (_,BX)],
        [inroom (BX,RX),inroom (robot,RX),nextto (robot,BX)],
        [],
        [inroom (BX,RX)]

open (DX) {Open door DX}
        [nextto (robot,DX),type (DX,door),status (DX,closed)],
        [status (DX,closed)],
        [status (DX,open)],
        [],
        []

close (DX) {Close door DX}
        [nextto (robot,DX),type (DX,door),status (DX,open)],
        [status (DX,open)],
        [status (DX,closed)],
        [],
        []
```

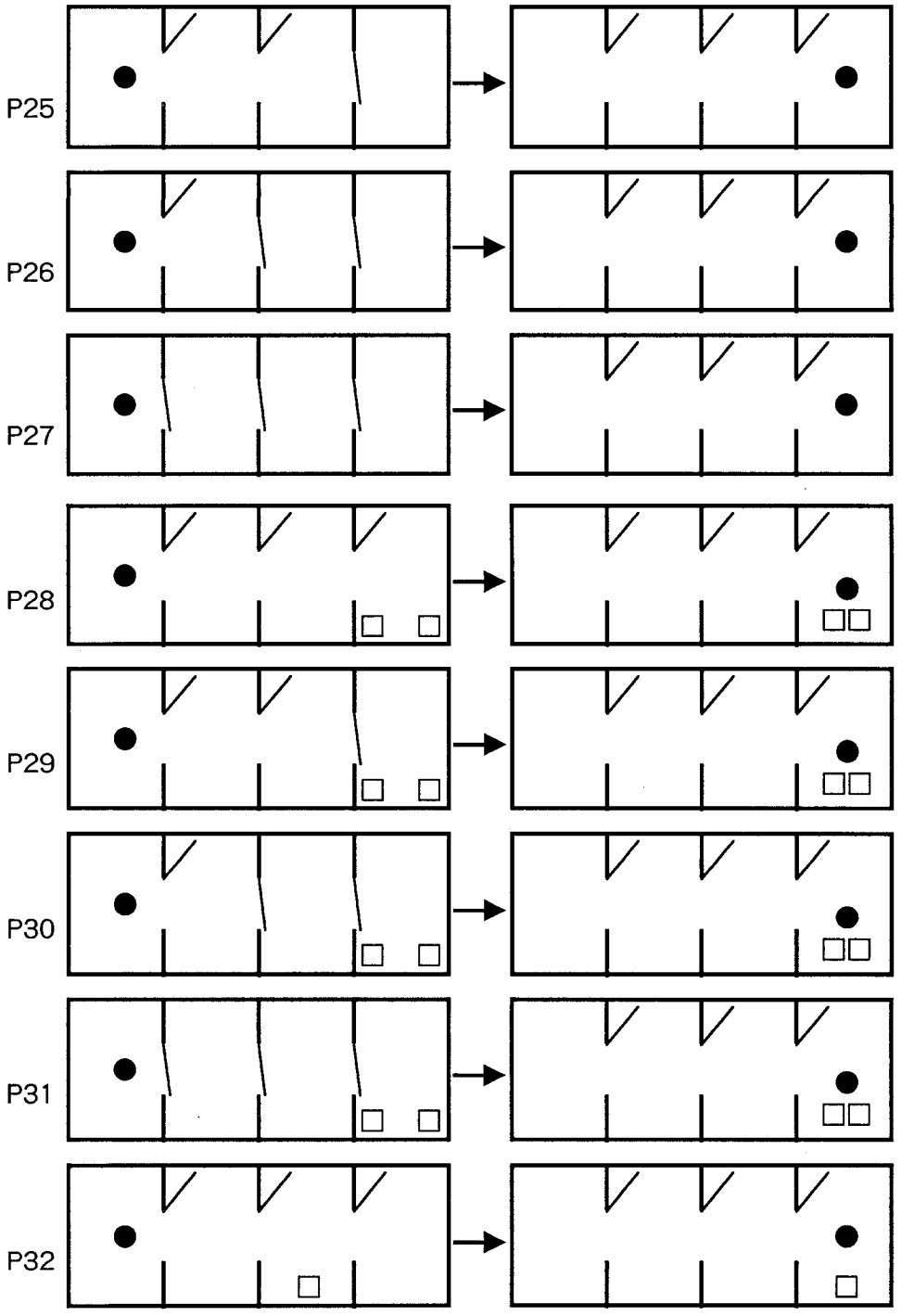
< Appendix7 > PiL2の訓練例

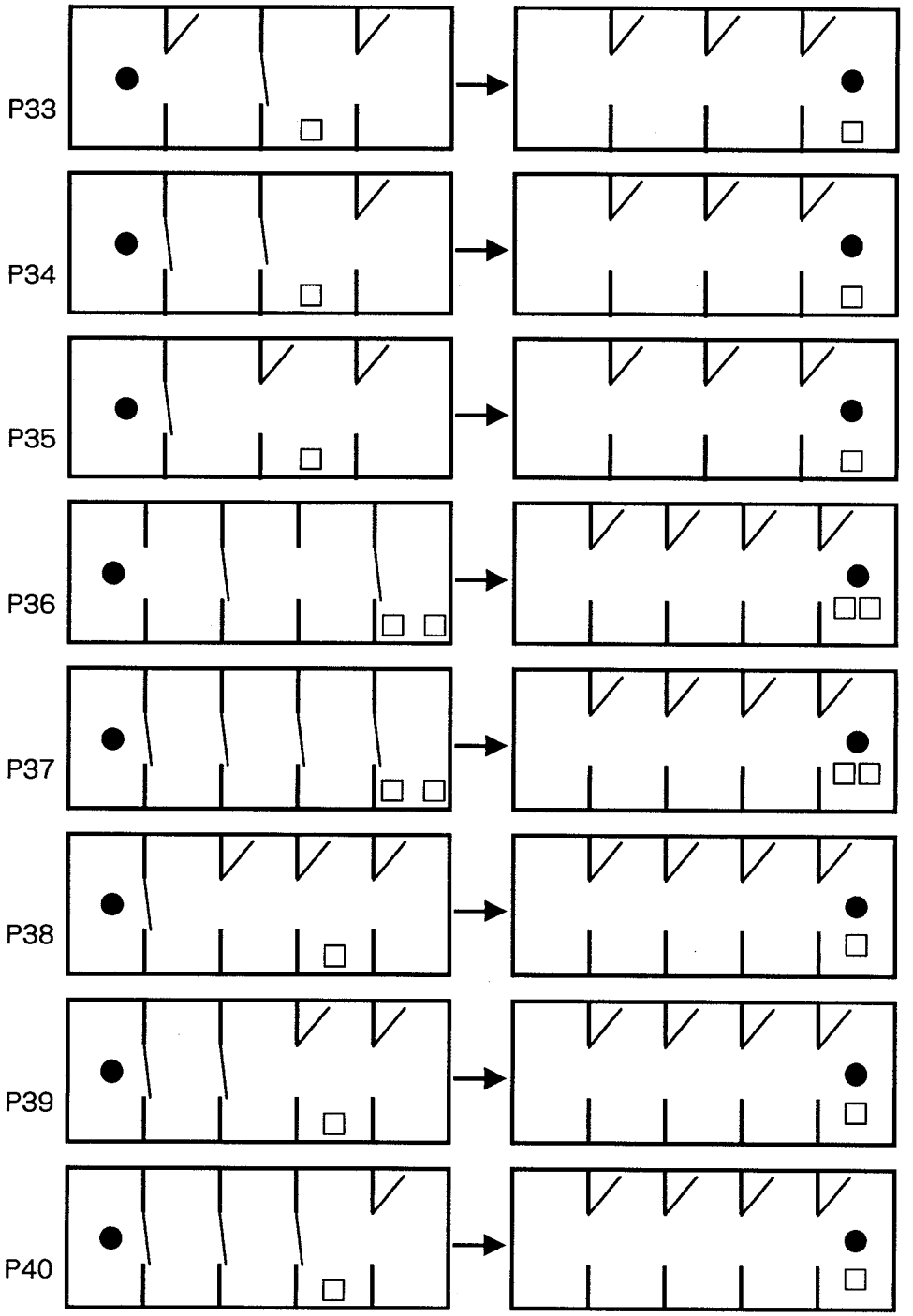
黒丸はロボット

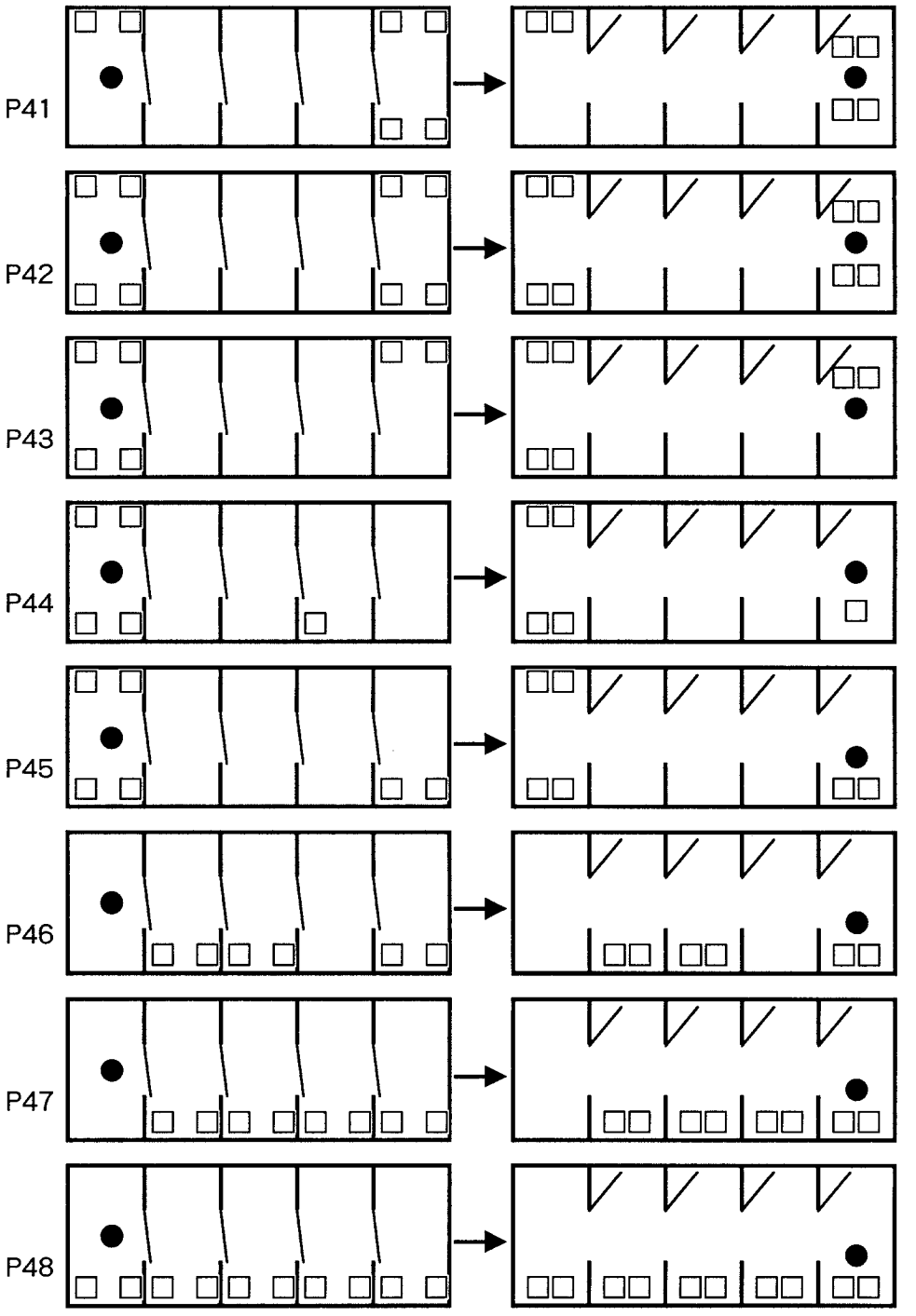


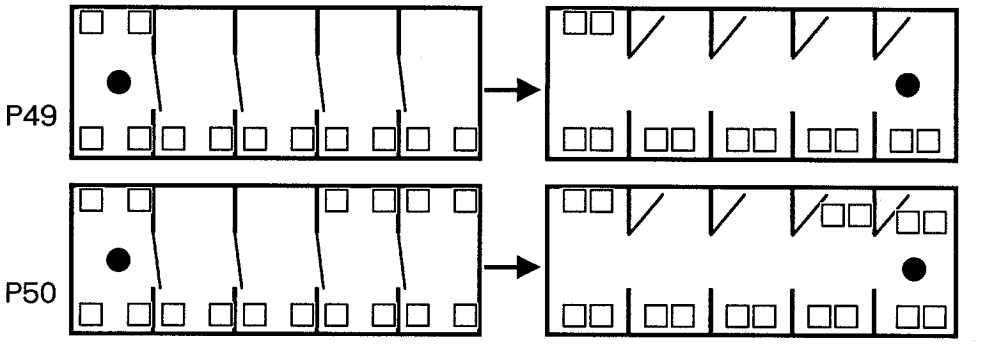

P9
P10
P11
P12
P13
P14
P15
P16

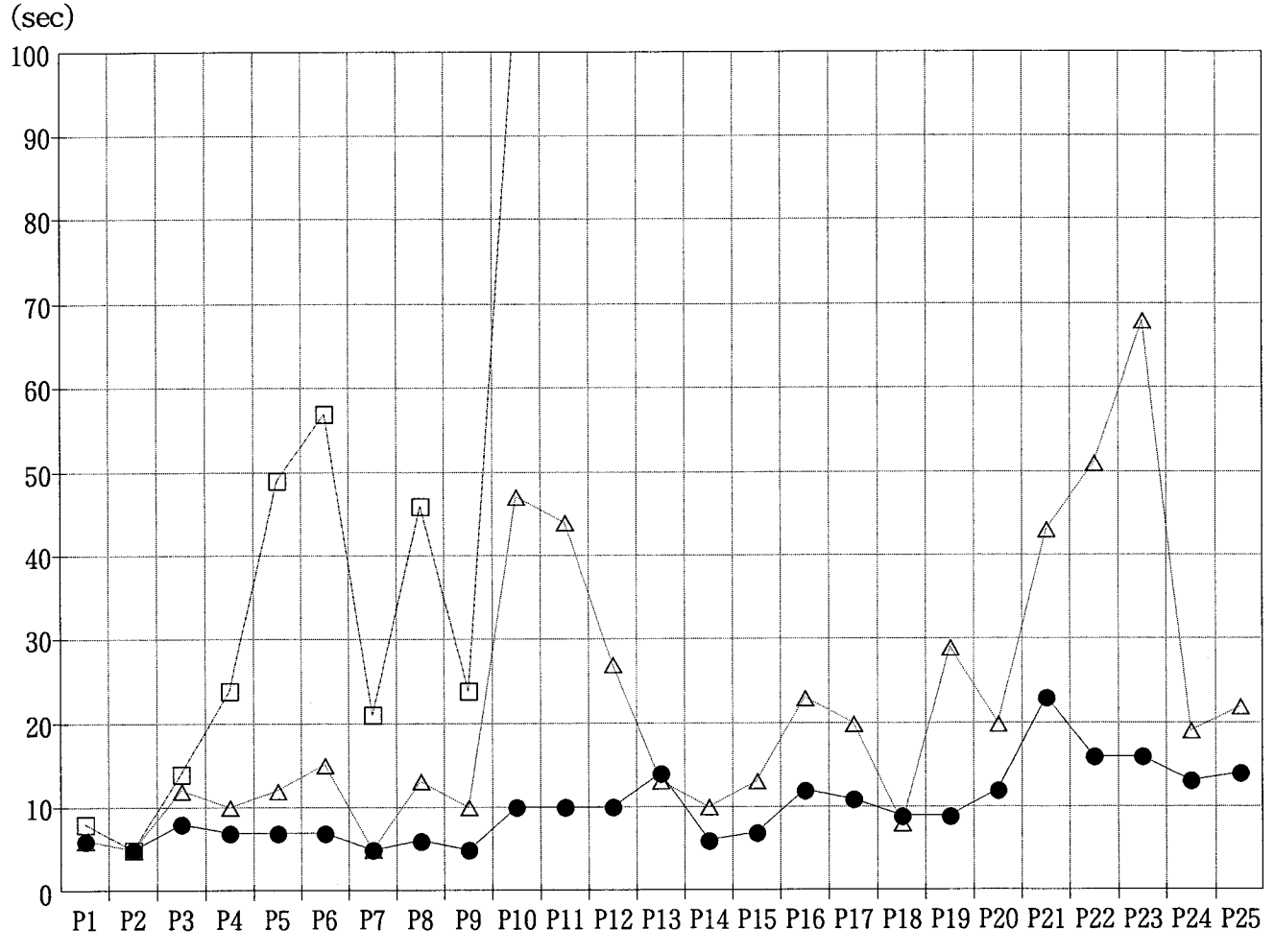

P17
P18
P19
P20
P21
P22
P23
P24


P25
P26
P27
P28
P29
P30
P31
P32


P33
P34
P35
P36
P37
P38
P39
P40

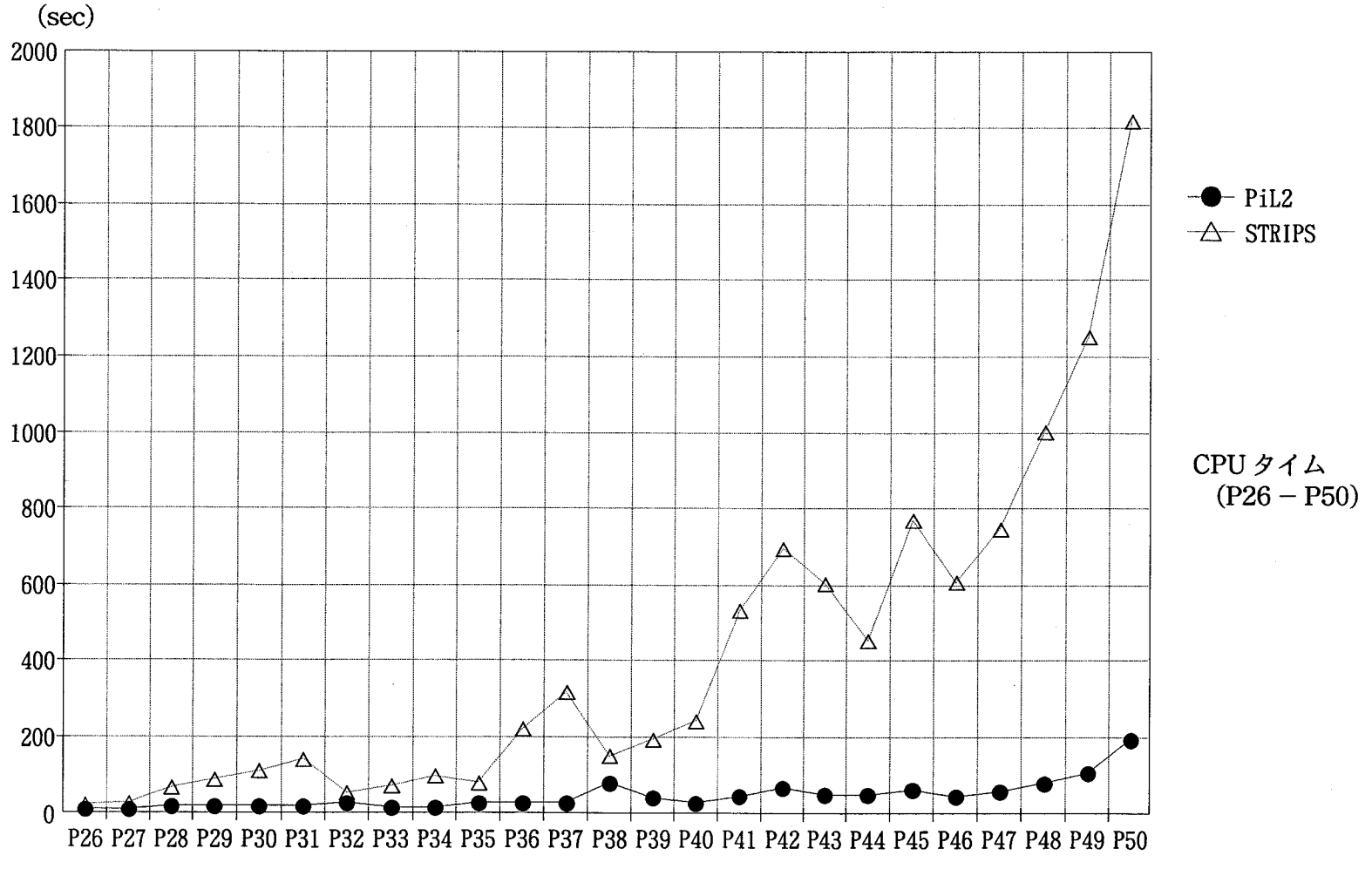

P41
P42
P43
P44
P45
P46
P47
P48


P49
P50

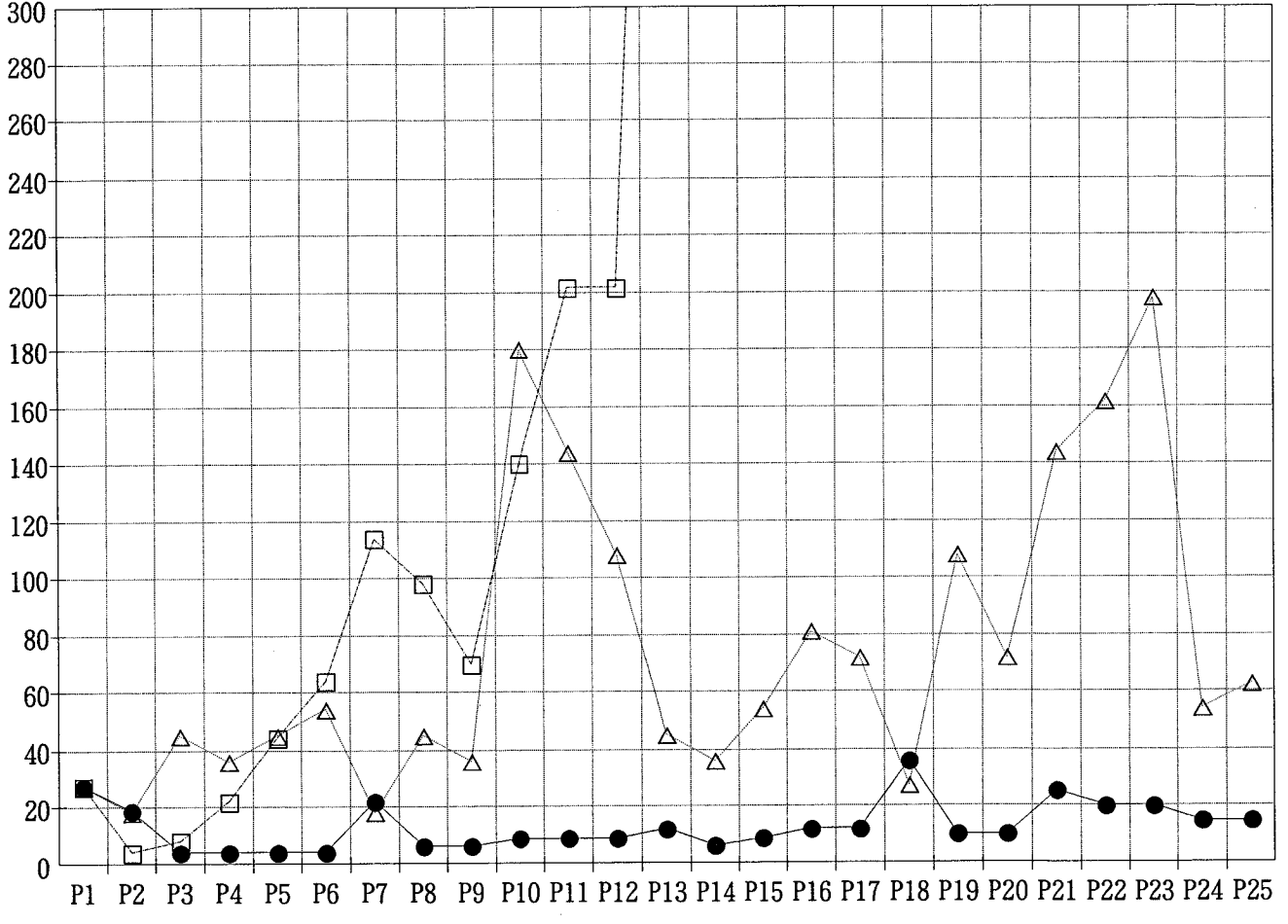


< Appendix8 >

CPU タイム
(P1 − P25)

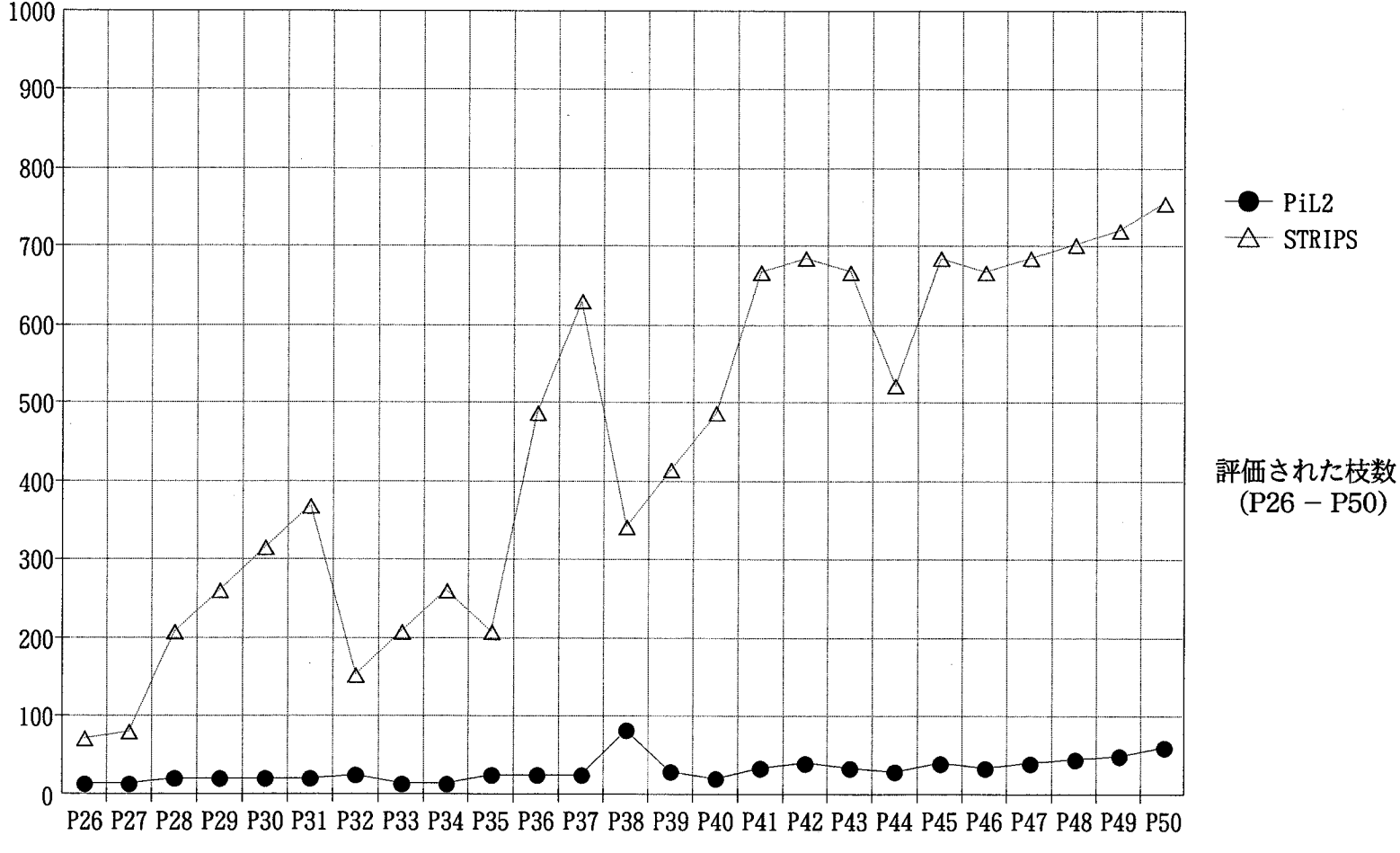


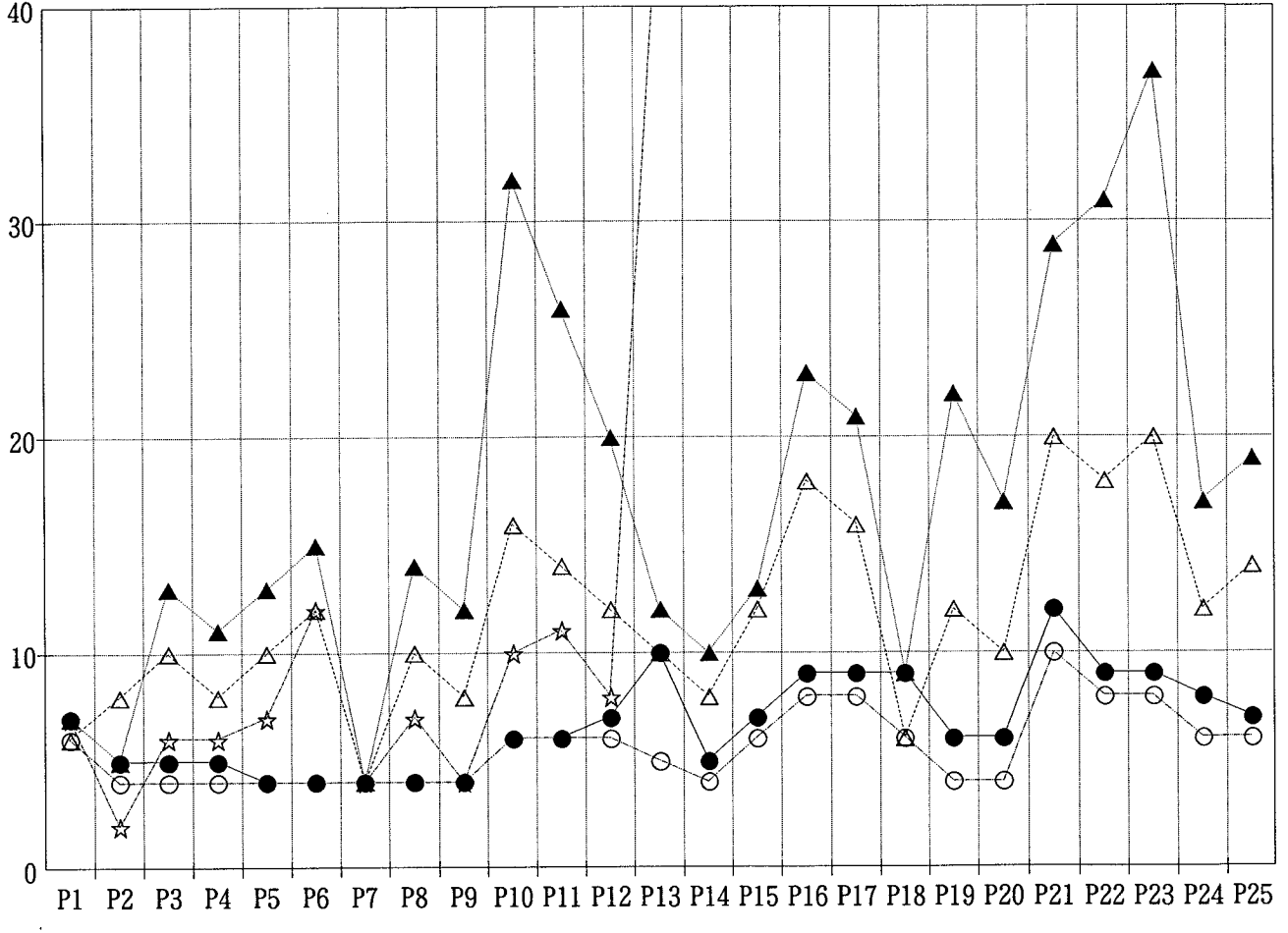
CPU タイム
(P26 − P50)



< Appendix9 >

評価された枝数
(P1 – P25)


1000
900
800
700
600
500
400
300
200
100
0
P26 P27 P28 P29 P30 P31 P32 P33 P34 P35 P36 P37 P38 P39 P40 P41 P42 P43 P44 P45 P46 P47 P48 P49 P50
PiL2
STRIPS
評価された枝数
(P26 − P50)



< Appendix10 >

展開されたノード数

PiL2

STRIPS

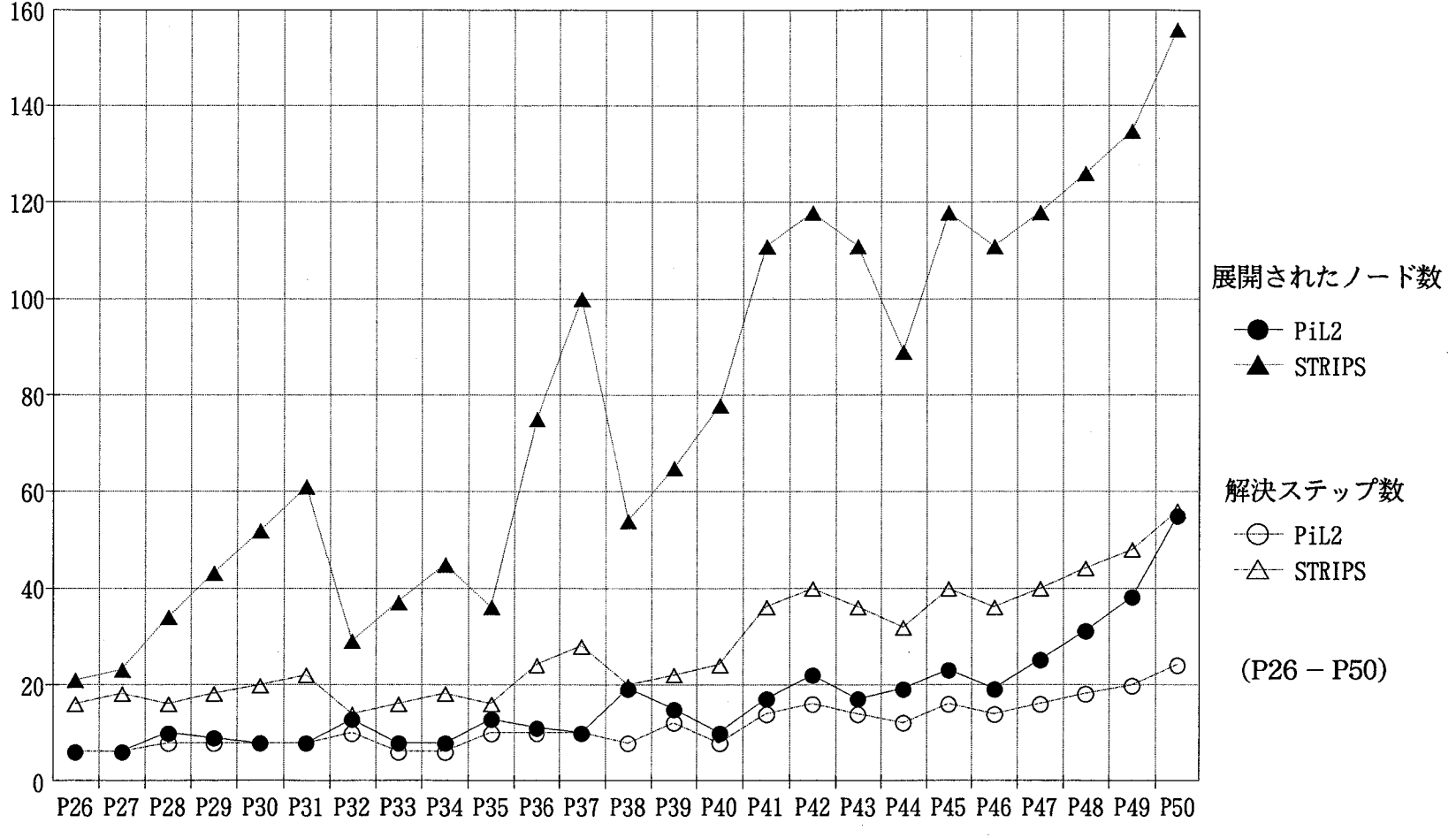
M-STRIPS

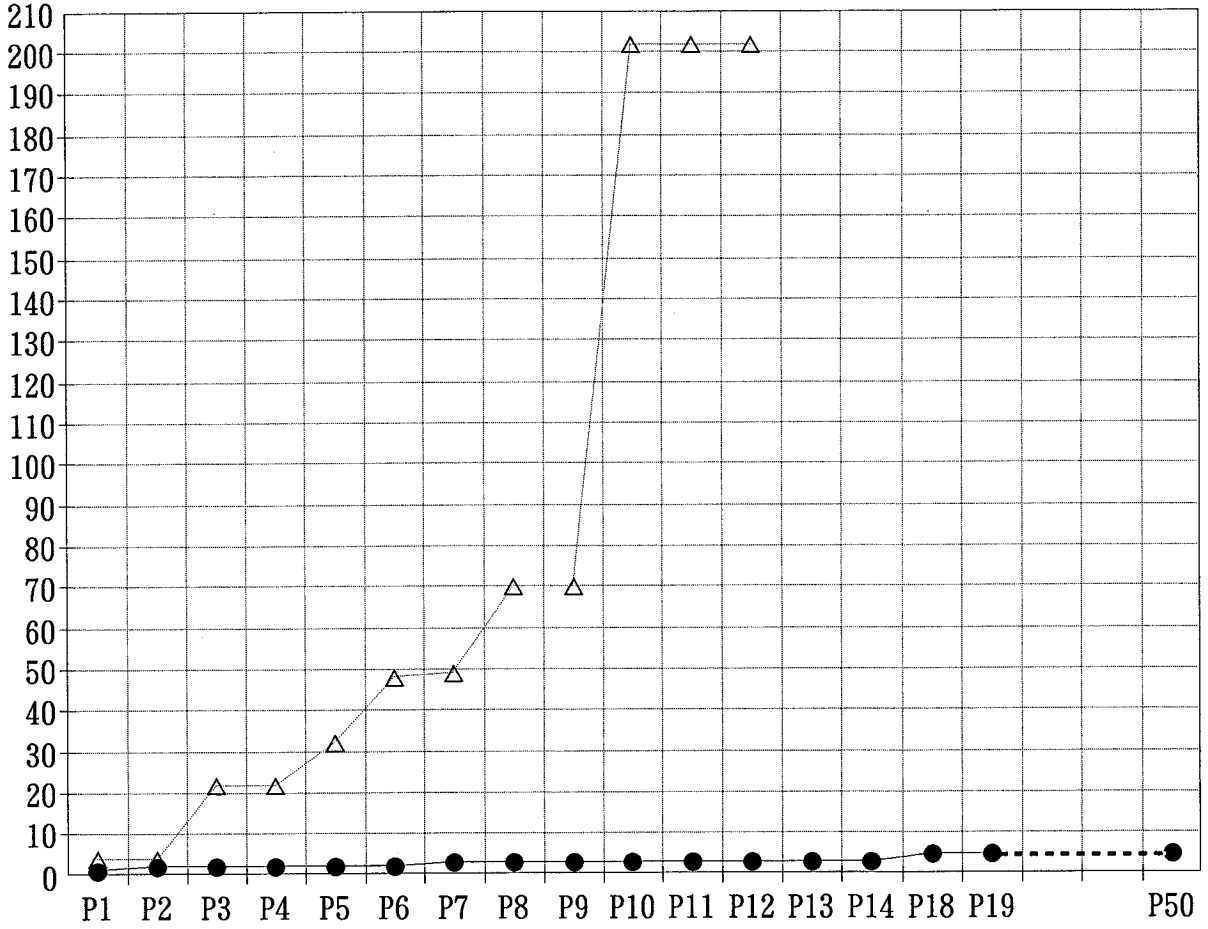
解決ステップ数

PiL2

STRIPS

(P1 − P25)


160
140
120
100
80
60
40
20
0
P26 P27 P28 P29 P30 P31 P32 P33 P34 P35 P36 P37 P38 P39 P40 P41 P42 P43 P44 P45 P46 P47 P48 P49 P50
展開されたノード数
PiL2
STRIPS
解決ステップ数
PiL2
STRIPS
(P26 − P50)



PiL2

P1 – P2 : 1
P3 – P6 : 2
P7 – P17 : 3
P18 – P50 : 5

M-STRIPS

< Appendix11 >

マクロオペレータの個数