人工知能学会研究会資料
SIG-AGI-010-02

# 単一化の機構を利用した階層型強化学習のテーブル圧縮手法の検討

# A Preliminary Study of a Table Compression Method Using a Unification Mechanism for a Hierarchical Reinforcement Learning

一杉裕志 [1*]　高橋直人 [1]　中田秀基 [1]　佐野崇 [2]
Yuuji Ichisugi[1]　Naoto Takahashi [1]　Hidemoto Nakada[1]　Takashi Sano[2]

[1] 産業技術総合研究所 人工知能研究センター
[1] National Institute of Advanced Industrial Science and Technology (AIST), AIRC
[2] 成蹊大学 理工学部 情報科学科
[2] Department of Computer and Information Science, Faculty of Science and Technology, Seikei University

**Abstract:** We propose a method of compressing the size of the table of the action value function for the hierarchical reinforcement learning architecture RGoal, using a mechanism of unification. Moreover, we study the characteristics of RGoal as a program synthesis system.

## 概要

我々は階層型強化学習アーキテクチャ RGoal を核として、ヒトの前頭前野周辺の情報処理機構のモデルを構築することを目指している。RGoal の行動価値関数のテーブルを素朴に実装するとサイズが大きくなり、学習速度と汎化性能が悪くなるという問題がある。そこで、単一化の機構を用いてテーブルを圧縮する手法を提案する。また、テーブルが理想的に圧縮された場合に知識がどのような表現になるかを記述する記述実験を行うことで、提案手法の有効性の予備的検討を行う。RGoal は強化学習を用いたプログラム合成システムと見なせるが、合成の対象となる「プログラミング言語」の設計は、性能に大きく影響する。そこで記述実験で記述したテーブルをプログラムと見なし、その性質について考察する。

## 1　はじめに

人間は何か目的を達成するために適切なサブゴールを設定できる。さらに必要に応じてそのサブゴールを再帰的に設定することもでき、その再帰の深さには制約がないように見える。この振る舞いにヒントを得た階層型強化学習のアーキテクチャとして、我々は RGoal アーキテクチャを提案した [14]。RGoal では、エージェントによるサブゴール設定は、プログラミング言語におけるサブルーチン呼び出しと似た振る舞いをする。我々はこの RGoal の機能を拡張していき、ヒトの前頭前野周辺の情報処理を再現し、やがては汎用人工知能を実現するための中核技術とすることを目指している。

多層の階層型強化学習のアーキテクチャの 1 つ MAXQ [2] は、以下の利点を持っている。

1. Subtask sharing: サブルーチンを異なるタスク間で共有して学習することで、マルチタスク環境での学習を速くできる。
2. Temporal abstraction: サブルーチンの組み合わせ方のみを探索することで探索空間を小さくし、高レベルのタスクの学習を速くできる。
3. State abstraction: サブルーチンの実行に差し支えない程度に状態を抽象化することで行動価値関数のパラメタの数を小さくし、サブルーチンの学習を速くできる。

Option-Critic アーキテクチャ[10] のような 2 層の階層型強化学習の場合は、 1 の Subtask sharing は下位層の学習、2 の Temporal abstraction は上位層の学習を、それぞれ加速する。また、ニューラルネットによる行動価値関数の関数近似は、おそらく 3 をある程度自律的に達成する。

我々が提案した RGoal アーキテクチャはこれまで 1 と 2 の利点については実現したが、3 のテーブルサイズの圧縮については取り組んでいなかった。そこで本

*連絡先：産業技術総合研究所
茨城県つくば市梅園 1 − 1 − 1 中央第 1
E-mail: y-ichisugi@aist.go.jp

論文では、単一化を使ったテーブル圧縮手法を提案し、理想的に圧縮された場合に知識がどのような表現になるかを記述する記述実験を行うことで、その有効性を予備的に検証する。理想的に圧縮された表現について知ることは、将来教師なし学習によって圧縮を実現する際のモデルおよび目的関数の設計に指針を与えるはずである。なお、圧縮手法の設計にあたっては、我々がこれまで取り組んできた大脳皮質モデル BESOM に関連した知見 [9][13] を強く踏まえている。

本研究のもう 1 つの動機は、 RGoal アーキテクチャを「強化学習によるプログラム合成システム」と見なし、合成されるプログラムの表現力や合成の効率に関する問題点を洗い出すことにある。強化学習によるプログラム合成は汎用人工知能実現に向けた有望なアプローチの 1 つである。例えば、AIXI[4] はチューリングマシンのプログラムを、UCAI[12] はより一般的な文法を持った言語のプログラムを強化学習で合成する汎用人工知能の理論である。現実的な性能で動作するシステムを目指す場合、どのようなプログラミング言語を用いるのかが、エージェントの学習速度・汎化能力・計算コストなどを決める重要なポイントとなる。ヒトの脳が強化学習でプログラム合成を行っているとしたら、用いている「プログラミング言語」は、人間が手で書くことに最適化されたプログラミング言語とは違い、強化学習による獲得に適したものであるはずである。また、それは生物が生きていくために解くべきタスクを解きやすいという性質を持ち、さらに脳の中の神経回路で実現可能なほどシンプルでなければならない。そこで理想的に圧縮されたテーブルをプログラムと見なし、これらの性質を満たしているかを考察する。

この論文は以下のような構成になっている。まず 2 章で RGoal アーキテクチャ、3 章で単一化について簡単に説明した後、4 章で提案手法について説明する。5 章で記述実験とその結果の考察を述べる。6 章では関連研究について述べ、7 章でまとめを行う。

# 2 RGoal アーキテクチャ

## 2.1 行動価値関数

我々が提案した RGoal アーキテクチャ[14] について簡単に説明する。

エージェントは各ステップごとに、行動するかサブルーチン呼び出しを行うかのどちらかを選択する。すべてのサブルーチンは通常のプログラミング言語のサブルーチンと同様に、相互に再帰呼び出しが可能である。

ここでは**サブルーチン** $g$ を、「任意の環境の状態からある 1 つの状態（サブゴール） $g$ に向かう方策」と定義する。したがって、サブルーチン $g$ が終了した時の環境の状態は $g$ に一意に決まる。（Option-Critic アーキテクチャ[10] などではサブルーチンの終了時の環境の状態は一意に決まらない。終了条件は学習によって獲得される。RGoal では、終了条件はセンサー入力から得られる外界の特徴量の 1 つだと考える。終了条件は強化学習アーキテクチャの本体によってではなく、特徴抽出の機構によって獲得されるものと想定している。）

RGoal アーキテクチャでは行動価値関数を $Q(s,a)$ ではなく $M(s,g,a)$ と表現する。$M(s,g,a)$ は状態 $s$ において行動 $a$ を取ったときの、状態 $s$ からサブゴール $g$ に到着するまでの間の報酬の総和の期待値である。

行動選択の式は通常の強化学習のものと似ており、例えばテーブル $M$ のもとでグリーディーに行動を選択する場合は以下のようにする。

$$a' \quad = \quad \underset{a}{\operatorname{argmax}}\, M(s,g,a) \tag{1}$$

テーブル $M$ は Sarsa などの通常の強化学習アルゴリズムを応用して学習することができる。

テーブル $M(s,g,a)$ のインデックスの次元は状態 $s$ の次元の 3 倍になり、素朴に実装するとテーブルのサイズが大きくなるという問題がある。

## 2.2 スタックの導入

我々が [14] で提案した RGoal アーキテクチャはスタックを用いず、フラットなテーブル $M$ と効率的でシンプルな操作の繰り返しで実現される。

しかし、5.7 節で述べるように、今回の記述実験の過程で、スタックがないとサブルーチンの再利用性が悪くなることが明らかになった。人間の脳内にデジタル計算機のように正確に動作し、かつ深さが無限のスタックがあるとは考えにくい。しかし、心理学において展望記憶 [5] と呼ばれている機構は、スタックの代替として動作し得る。実際の脳がどのようにしてスタックを代替しているかは現時点では不明なので、今回は神経科学的妥当性にはこだわらず、計算機上で素直に実装できる方法でスタックをアルゴリズムに組み込んだ。詳しくは 5.1 節で述べる。

# 3 単一化

**単一化** (unification) は数理論理学や計算機科学でよく使われる操作である。確定値と不確定値の組み合わせで構成される 2 つのデータ構造を単一化すると、不確定だった値が確定するか、あるいは取り得る値に関する制約が強められる。例えば X, Y を不確定値を表す変数としたとき、2 つの数値ベクトル (2, X), (Y, 3) を単一化すると (2,3) という数値ベクトルになり、不

| インデックス | 値 |
|---|---|
| 0 | 1.0 |
| 1 | 3.0 |
| 2 | 1.0 |
| 3 | 1.0 |
| 4 | 1.0 |

→

| パターン | 値 |
|---|---|
| 1 | 3.0 |
| X | 1.0 |

図 1: 圧縮の対象となるテーブルの例。インデックスが 1 の時の値が 3.0 でその他の時が 1.0 。右のように 2 つのルールで圧縮表現できる。各ルールはパターンと値の組からなる。

| X \ Y | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 2.0 | 1.0 | 1.0 | 3.0 | 1.0 |
| 1 | 1.0 | 2.0 | 1.0 | 3.0 | 1.0 |
| 2 | 1.0 | 1.0 | 2.0 | 3.0 | 1.0 |
| 3 | 1.0 | 1.0 | 1.0 | 4.0 | 1.0 |
| 4 | 1.0 | 1.0 | 1.0 | 3.0 | 2.0 |

→

| パターン | 値 |
|---|---|
| (3,3) | 4.0 |
| (X,3) | 3.0 |
| (X,X) | 2.0 |
| (X,Y) | 1.0 |

図 2: 2 次元のテーブルの例。インデックスの組 $(X, Y)$ のパターンを使って右のように 4 つのルールに圧縮できる。

確定だった変数への値の割り当てが X=3, Y=2 に確定する。 2 つのデータ構造を同一にする変数割り当てが存在しない場合は、「単一化に失敗する」と言う。例えば (2,X) と (3,Y) は単一化に失敗する。

2 つのベクトルの単一化の機構はベイジアンネットで実現可能である [13]。

理論言語学における文法記述の枠組みの多く（例えば CCG[3]）が単一化文法である。単一化文法とは、単一化を使って構文解析を進めていく文法枠組みの総称である。自然言語の性質を表現する道具として洗練され到達した 1 つの結果が単一化文法であるという事実は、脳の言語野で何らかの形で単一化が行われていることを強く示唆している。前頭前野も言語野と同じ大脳皮質であるため、前頭前野もまた単一化を行っている可能性があると類推できる。

本研究で単一化を用いるもう 1 つの重要な動機として、人間が行う記号処理的な思考の再現という目標がある。本稿で提案する機構は、まだ単純かつ不完全ではあるものの、記号処理と統計的機械学習の統合に大きく近づいたものになっている。

# 4 テーブル圧縮

## 4.1 パターンマッチによるテーブル圧縮

提案手法では、テーブルのインデックスの代わりにパターンを用い、パターンマッチによりテーブルの要素を選択する。パターンの処理に多少コストはかかるものの、強化学習で学習すべきパラメタの数が大幅に減れば、汎化性能の向上につながると思われる。

| X \ Y | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | −∞ | 1.0 | −∞ | −∞ | −∞ |
| 1 | −∞ | 1.5 | −∞ | −∞ | −∞ |
| 2 | −∞ | 1.2 | −∞ | −∞ | −∞ |
| 3 | −∞ | 2.0 | −∞ | −∞ | −∞ |
| 4 | −∞ | 1.3 | −∞ | −∞ | −∞ |

→

| パターン | 値 |
|---|---|
| (X,1) | 1.4 |
| (X,Y) | −∞ |

図 3: 値の精度を落とすことでより小さく圧縮できる例。厳密に表現するには 5 つのルールが必要だが、精度を落とせば 2 つのルールに圧縮できる。

| パターン | 値 |
|---|---|
| $(p_{11}, p_{12}, p_{13})$ | $v_1$ |
| $(p_{21}, p_{22}, p_{23})$ | $v_2$ |
| ... | ... |
| $(S, G, A)$ | $v_{default}$ |

図 4: RGoal の行動価値関数のテーブル $M(s, g, a)$ をルールの集合で表現したもの。

図 1 左は、インデックスが 1 の時の値は 3.0 、残りは 1.0 のテーブルである。提案手法では、これを図の右のように、パターンと値のペアからなる**ルール** 2 つで表現する。図 2 は 2 次元のテーブルの例で、インデックスの組 $(X, Y)$ のパターンを使って図の右のように 4 つのルールに圧縮できる。

なお、ルールの順序には意味はないものとする。値がパターンにマッチするルールが複数ある場合、最も特殊なパターンをもつルールの方を選択する。ここで、「最も特殊なパターン」とは、用いる変数の種類が最も少ないパターンと定義する。例えば値 $(2, 2)$ はパターン $(X, X)$ と $(X, Y)$ の両方にマッチするが、パターン $(X, X)$ の方は 1 種類の変数のみを用いているためこれが選択される。

これまで述べた 2 つの例は圧縮によって値の精度が落ちることはないが、一般に精度を落としてよければより小さく圧縮することができる。例えば図 3 の左のテーブルを厳密に表現するには 5 つのルールが必要だが、精度を落とせば図の右のように 2 つのルールで表現できる。

## 4.2 RGoal における行動選択の方法

この圧縮方法を用いた場合の RGoal における行動選択の方法について説明する。まず RGoal の行動価値関数 $M(s, g, a)$ を図 4 のように、 3 つ組 $(s, g, a)$ に対

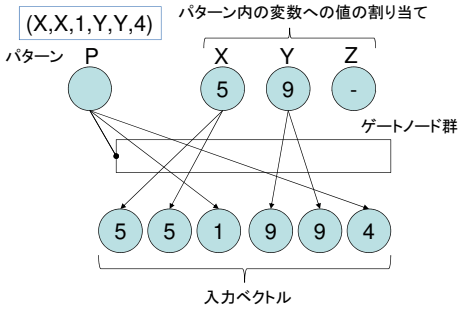


図 5: ベイジアンネットによるパターンマッチを行う回路。左上のノード P はパターンの ID を値として持つ。ノード P は ID に対応するパターンに応じて、上の層と下の層の間の結合をゲートノード [9][11] を使って適宜制御し、同じ値になるべきノードどうしを結合する。下の層のみに観測値を与えて事後確率最大の値の組を推論すると、入力にマッチするパターンと変数割り当てが上の層の値として表現される。

するパターンとその時の $M(s,g,a)$ の値の組からなるルールの集合として表現する。エージェントは、現在の状態 $s$ とサブゴール $g$ が与えられたとき、パターン $(s,g,A)$ と単一化できるパターンを持つルールを検索し、単一化の結果、変数 $A$ に割り当てられた行動を選択する。マッチする「最も特殊なパターン」が複数ある場合は、その中でもっとも価値の高い $A$ を選択する。

### 4.3 ベイジアンネットでの実現方法

テーブルの圧縮は一種の関数近似であり、学習データが十分にあれば深層学習でルールを獲得可能であろう。しかし、ヒトの知能のように極端に少ない学習データからのルール獲得を目指す場合、その目的に特化した機械学習モデルの設計が必要である。少数の学習データから適切にルールを獲得できれば、学習データにない値の組み合わせに対しても汎化することが期待できる。

この目的に特化した機械学習モデルの候補として、ベイジアンネットを用いたパターンマッチの機構の 1 つの案を図 5 に示す。パターンの集合の学習が終わったネットワークの下の層に入力ベクトルを与えれば、上の層に入力にマッチするパターンと割り当てられた変数の値が推論される。

最も特殊なパターンを選択する機構は、上の層右側のパターン変数を表すノード群の事前分布を適宜設定し、使う変数の数が少ないほど全体の同時確率が大きくなるようにすることで実現できると考えている。

このベイジアンネットは noisy-OR ゲートや NOT などの組み合わせのみで構成可能であり、パラメタ数が爆発することはない [9]。

パターンにマッチするルールのうち最も価値の高いものを選択する機構は、別途別の機構 [7] と組み合わせることで実現する。

このネットワークはオートエンコーダーになっており、原理的には下の層に学習データを与えることで、上の層に圧縮されたパターンを教師なしで獲得できるはずである。ただし、学習を成功させるためにはパラメタに適切な事前分布を設定する必要があるだろう。

大脳皮質においては、1 つのルールが 1 つのミニコラムに対応するのではないかと考えている。仮にヒトの大脳皮質全体のニューロン数を１００億個とし、その 1/100 がルールを表現する領野で、1 つのミニコラム内のニューロン数が１００個だとすると、１００万個のルールが保持されることになる。

## 5 プログラムの記述実験

### 5.1 プログラムの記述方法とそのインタープリタ

前章で述べた方法で圧縮された行動価値関数のテーブルは、エージェントの行動を決定するプログラムと見なすこともできる。この「プログラミング言語」の性質を知るために、まずはプロトタイピングとして、テーブルが理想的に圧縮された場合のプログラムを直接手で記述する記述実験を行った。書かれたプログラムは、単一化と探索の機能を持つ prolog 言語で実装されたインタープリタによって動作確認した。

プログラム（テーブル）の記述の方法を以下に説明する。圧縮の結果得られるであろうルールのうち、パターンの部分だけを、“m$(s,g,a)$.” という prolog 言語の fact （body のないホーン節）の形で列挙していく。今回は各パターンに対応する値は書かず、書かれたパターンに対応する値はすべて「$-\infty$ ではない値」と解釈する。一方、書かれているどのパターンにもマッチしない場合はデフォルトの値 $-\infty$ になるものとする。

行動 $a$ には、プリミティブな行動の場合は識別子 ($right, left$ など)、サブルーチン $m$ の呼び出しの場合は “g($m$)” を書く。サブルーチン $m$ を呼び出すと現在のサブゴール $g$ はスタックに退避され、状態 $s$ が $m$ に到達したときに復元される。

エージェントの行動によって $s \rightarrow m1 \rightarrow m2 \rightarrow m3 \rightarrow g$ というふうに環境の状態を遷移させたい場合は、

```
m(s,g,g(m1)).
m(m1,g,g(m2)).
m(m2,g,g(m3)).
```

というふうに、サブルーチンを呼び出すルールを列挙すればよい。。このプログラムが実行可能になるため

```
1: procedure EPISODE(s, g)
2:     stack ← empty
3:     Choose a which matches m(s, g, a)
4:     while not (stack is empty and a = ret) do
5:         # Take action.
6:         if a is ret then
7:             s' ← s; g' = stack.pop()
8:         else if a is g(m) then
9:             stack.push(g)
10:            s' ← s; g' ← m
11:        else
12:            Take action a, observe s'
13:            g' ← g
14:        # Choose action.
15:        if s' = g' then
16:            a' ← ret
17:        else
18:            Choose a' which matches m(s', g', a')
19:        s ← s'; g ← g'; a ← a'
20:    return
```

図 6: 今回の記述実験に用いたエージェントの行動アルゴリズムの疑似コード。我々が提案した RGoal のアルゴリズム [14] から思考モードと呼ぶ機能と学習機能をなくして簡略化した上で、サブルーチン呼び出しのためのスタックを追加した。EPISODE$(s, g)$ は開始状態 s からゴール g に到達可能な行動系列を非決定論的に 1 つ選んで出力する。ルール $m(s, g, a)$ の集合をプログラムと見なす場合、これはプログラムのインタープリタに相当する。

```
episode(S,G,Log) :- episodeN(S,G,0,Log), nl.
episodeN(S,G,50,Log) :- !, fail.   % Max 50 steps
episodeN(S,G,N,Log) :- nl, print(N),
    chooseAction(S,G,A),
    mainLoop(S,G,A,[],0,N,Log).
episodeN(S,G,N,Log) :- N1 is N+1, episodeN(S,G,N1,Log).

mainLoop(S,G,ret,[],I,N,[[S,G,ret]]) :- I=N, !.
mainLoop(S,G,ret,[],I,N,Log) :- !, fail.
mainLoop(S,G,A,Stack,I,N,Log) :- I=N, !, fail.
mainLoop(S,G,A,Stack,I,N,[[S,G,A]|Log]) :- print([I]),
    takeAction(S,G,A,S1,G1,Stack,Stack1),
    chooseAction(S1,G1,A1),
    I1 is I+1,
    mainLoop(S1,G1,A1,Stack1,I1,N,Log).

chooseAction(S,G,A) :- S=G, !, A=ret.
chooseAction(S,G,A) :- m(S,G,A).

takeAction(S,G,A,S1,G1,[GG|Stack],Stack)
    :- A=ret, !, S1=S, G1=GG.
takeAction(S,G,A,S1,G1,Stack,[G|Stack])
    :- A=g(M), !, S1=S, G1=M.
takeAction(S,G,A,S1,G1,Stack,Stack)
    :- e(S,S1,A), G1=G.
```

図 7: インタープリタ（図 6）の prolog による実装。

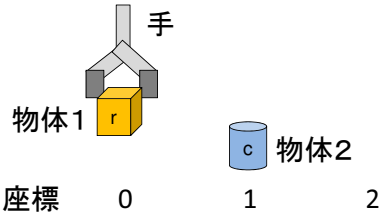


図 8: 単純化した積み木の世界。物体の座標は 0,1,2 のいずれか。この絵の状況の時、環境の状態は 6 次元ベクトルの値 $(H, P0, O1, P1, O2, P2) = (hld1, 0, r, 0, c, 1)$ で表現される。ここで $hld1$ は手が物体 1 をつかんでいる状態、 $r$ は直方体、 $c$ は円柱を表す値とする。（詳しくは本文参照。）

には、呼び出されるサブルーチンも実装されている必要がある。つまり、$s \to m1$, $m1 \to m2$, $m2 \to m3$, $m3 \to g$ という各状態遷移も実行可能になるようにプログラムされている必要がある。なお、最後の $m3 \to g$ という状態遷移を起こすサブルーチン呼び出しを書く必要はない点に注意されたい。状態が $m3$ に到達した時点でサブゴールの値 $g$ がスタックから戻され、自動的に $m3 \to g$ が実行される。

普通のプログラミング言語のサブルーチンと大きく違うのは、入り口と出口である。この「プログラミング言語」ではサブルーチンの出口だけが決まっており、潜在的に任意の状態が入り口になり得る。

今回の記述実験に用いたインタープリタの疑似コードを図 6 に、 prolog による実際の実装コードを図 7 に示す。この実装では「最も特殊なパターン」ではなくマッチするパターンのうちどれか 1 つを選択する実装になっている。また、深さ優先探索では無限ループになる場合があるので、幅優先で探索するようにしている。

## 5.2 環境：単純化した積み木の世界

記述実験では、単純化した積み木の世界（図 8）におけるいくつかのタスクを解くルールをプログラムしていく。

環境の状態は手、物体 1 、物体 2 それぞれの形と位置のペアで表現する。この表現は視覚的短期記憶に関する Kahneman らの仮説 [1] が想定するオブジェクトファイルの形になっている。エージェントは自分の周りにある重要な物体にあらかじめ逐次的に注意を向けることで、このような内部表現を構築済みであると仮定する。（注意を向けていないすべての物体をエージェントは無視しており、この時点ですでに環境の状態は大幅に抽象化されている。）

手の形は $H \in \{open, hld1, hld2\}$ であり、それぞれ手

が開いた状態、物体 1 をつかんでいる状態、物体 2 をつかんでいる状態を表す。物体 1 、物体 2 の形は $O1, O2 \in \{r, c\}$ であり、$r$ は直方体 (rectangular parallelopiped) 、$c$ は円柱 (column) を表す値である。手、物体 1 、物体 2 の位置は $P0, P1, P2 \in \{0, 1, 2\}$ である。環境の状態は $(H, P0, O1, P1, O2, P2)$ という 6 次元ベクトルで表現する。

この環境において、プリミティブ行動を実行するルールの集合を図 9 に示す。これらはいわば基本ライブラリであり、様々なタスクを解く他のプログラムから利用される。

## 5.3 タスク 1 : 1 つの物体の移動

1 つ目のタスクは、物体 1 を、物体 2 がある場所に移動するタスクである。このタスクを解くプログラムは図 10 で定義され、そこから図 9 で定義されたプリミティブ行動の実行が適宜呼び出される。このプログラムの実行結果を図 11 に示す。

## 5.4 タスク 2 : 2 つの物体を同じ場所に移動

2 つ目のタスクは物体 1 と 2 を手がある位置に移動するタスクである。図 12 と図 13 はこのタスクを解くプログラムであり、その実行結果を図 14 に示す。これはスタックを使った 2 段階の深さのサブルーチン呼び出しの実行例になっている。

このタスクは最適解が 1 つに決まらない例である。このタスクを解くためには、物体 1 と 2 のどちらを先に動かしてもかまわず、実際にプログラムを実行すると、両方が解として得られる。プログラムがパターンを使って抽象的に書かれているため、このような実行順序の柔軟性が実現されている[1]。

## 5.5 テーブル圧縮についての評価

提案手法では、パターンを用いることでテーブルのサイズを大幅に圧縮できている。例えば下記の物体 1 を動かすルールは、これ 1 つで O1,O2,P2 のすべての値の組み合わせ 2 x 2 x 3 = 12 通りの状況に適用できる。

```
m([hld1,0,O1,0,O2,P2], [hld1,1,O1,1,O2,P2], right).
```

また、タスク 1 、タスク 2 を解くプログラムはどちらも P0,P1,P2,O1,O2 のすべての値の組み合わせ 3 x 3 x 3 x 2 x 2 = 108 通りの状況で動くように書かれており、汎用性があると言える。言うまでもなく現実世界では物体も位置もはるかに多くの値を取り得るため、パターンを使うことによる圧縮の効果は極めて大きいと思われる。

各サブルーチンの価値（実行コスト）の精度についてはどうだろうか。ここでは、サブルーチンの実行に必要なステップ数をコストと考える。図 9 のプリミティブを実行するサブルーチンに関しては、パターンで抽象化することによる精度の低下はない。しかし、図 10、図 12、図 13 のプログラムでは移動元と移動先が抽象化されており、実際の移動距離は実行ごとに異なるため、学習される値はその平均となり、精度は落ちる。

## 5.6 プログラミング言語の性質についての考察

汎用人工知能としてのプログラム合成に適したプログラミング言語は、下記の特徴を持っていることが望ましい。

1. 無意味なプログラムが生成されにくい。
2. 知識の再利用がしやすい。
3. 表現の小さな変化は意味を小さく変化させ、連続的な改良がしやすい。
4. プログラムと自然言語の間の相互変換がしやすい。
5. 何らかの意味で万能である。（チューリング完全など。）
6. 生物が生きていくために解くべきタスクを解きやすい。
7. 脳の中の神経回路で実現可能なほどシンプルである。

プログラミング言語としての RGoal はこれらの条件をよく満たしていると今のところは考えている。今後、デモプログラムによってそのことを示していきたいと考えている。

無意味なプログラムの例として、無限ループに陥るプログラムがある。実はランダムに初期化したテーブル $M(s, g, a)$ にもとづく方策を実行すると、容易に行動の無限ループが起きる。ところが、テーブルを強化学習で改善していくと、短いループは価値が速やかに下がり、エージェントはループから脱出する。つまり都合のよいことに、無限ループするプログラムが速やかにそうでないプログラムに変化するのである。

型エラーのあるプログラムも、無意味なプログラムの例である。今回の記述実験では変数に型はないが、単一化の機構は型チェックとは相性が良い。物体の種類を 1 つの変数ではなく、様々な操作の適用可能性を表す変数のベクトルで表現することで、型に相当するも

[1] 5.1 節で述べたように最後のサブルーチン呼び出しは不要なので図 12 の 2 つのルールのうちどちらかを削除してもプログラムは動く。その場合は、どちらかを先に動かす解しか得られない。

```
% 物体をつかむ。
m([open,P1,O1,P1,O2,P2], [hld1,P1,O1,P1,O2,P2], hold1).
m([open,P2,O1,P1,O2,P2], [hld2,P2,O1,P1,O2,P2], hold2).

% 物体をはなす。
m([hld1,P1,O1,P1,O2,P2], [open,P1,O1,P1,O2,P2], release).
m([hld2,P2,O1,P1,O2,P2], [open,P2,O1,P1,O2,P2], release).

% 物体を持っていない状態で手を移動する。
m([open,0,O1,P1,O2,P2], [open,1,O1,P1,O2,P2], right).
m([open,0,O1,P1,O2,P2], [open,2,O1,P1,O2,P2], right).
m([open,1,O1,P1,O2,P2], [open,0,O1,P1,O2,P2], left).
m([open,1,O1,P1,O2,P2], [open,2,O1,P1,O2,P2], right).
m([open,2,O1,P1,O2,P2], [open,1,O1,P1,O2,P2], left).
m([open,2,O1,P1,O2,P2], [open,0,O1,P1,O2,P2], left).

% 物体 O1 をつかんだ状態で手を移動する。
m([hld1,0,O1,0,O2,P2], [hld1,1,O1,1,O2,P2], right).
m([hld1,0,O1,0,O2,P2], [hld1,2,O1,2,O2,P2], right).
m([hld1,1,O1,1,O2,P2], [hld1,0,O1,0,O2,P2], left).
m([hld1,1,O1,1,O2,P2], [hld1,2,O1,2,O2,P2], right).
m([hld1,2,O1,2,O2,P2], [hld1,1,O1,1,O2,P2], left).
m([hld1,2,O1,2,O2,P2], [hld1,0,O1,0,O2,P2], left).

% 物体 O2 をつかんだ状態で手を移動する。
m([hld2,0,O1,P1,O2,0], [hld2,1,O1,P1,O2,1], right).
...
```

図 9: プリミティブ行動を実行するルール。他のプログラムから基本ライブラリのように利用される。各行の “$\mathrm{m}(s, g, a)$.” は「現在の状態が $s$ でサブゴールが $g$ のとき、行動 $a$ の価値は $-\infty$ ではない」ということを意味する。大文字で始まる識別子は変数、小文字で始まる識別子や数値は定数である。手が物体を持っていない状態 (open) では、手を動かしても 2 つの物体は動かない。物体をつかんでいる状態 (hld1 または hld2) では、手とつかんでいる物体は同時に動く。

```
%   1. 手を P1 に移動する。
m([open,P0,O1,P1,O2,P2], [open,P2,O1,P2,O2,P2], g([open,P1,O1,P1,O2,P2])).
%   2. O1 をつかむ。
m([open,P1,O1,P1,O2,P2], [open,P2,O1,P2,O2,P2], g([hld1,P1,O1,P1,O2,P2])).
%   3. 手を P2 に移動する。
m([hld1,P1,O1,P1,O2,P2], [open,P2,O1,P2,O2,P2], g([hld1,P2,O1,P2,O2,P2])).
%   4. 手をはなす。
% 定義不要。
```

図 10: タスク 1（物体 O1 を場所 P1 から P2 に移動するタスク）を解くプログラム。3 つのサブルーチンを呼び出しているが、それらはいずれも図 9 で定義ずみである。なお、最終の手順「4. 手をはなす。」は図 9 で定義ずみのルールで実行されるので、サブルーチン呼び出しを書く必要はない。

```
?- episode([open,1,r,0,c,2], [open,2,r,2,c,2],Log), printList(Log).

[[open,1,r,0,c,2],[open,2,r,2,c,2],g([open,0,r,0,c,2])]
[[open,1,r,0,c,2],[open,0,r,0,c,2],left]
[[open,0,r,0,c,2],[open,0,r,0,c,2],ret]
[[open,0,r,0,c,2],[open,2,r,2,c,2],g([hld1,0,r,0,c,2])]
[[open,0,r,0,c,2],[hld1,0,r,0,c,2],hold1]
[[hld1,0,r,0,c,2],[hld1,0,r,0,c,2],ret]
[[hld1,0,r,0,c,2],[open,2,r,2,c,2],g([hld1,2,r,2,c,2])]
[[hld1,0,r,0,c,2],[hld1,2,r,2,c,2],right]
[[hld1,1,r,1,c,2],[hld1,2,r,2,c,2],right]
[[hld1,2,r,2,c,2],[hld1,2,r,2,c,2],ret]
[[hld1,2,r,2,c,2],[open,2,r,2,c,2],release]
[[open,2,r,2,c,2],[open,2,r,2,c,2],ret]
```

図 11: タスク 1 の解の実行ログの例。最初手が座標 1 にあり、物体 1 を座標 0 から座標 2 に動かすというタスク。実行ステップごとに得られる [s,g,a] の具体的な値が実行ログとして出力されている。サブルーチン呼び出し g([...]) と復帰 ret を除いたプリミティブ行動の系列は left,hold1,right,right,release となり、確かにタスクを解く手順になっている。スタートとゴール状態における座標を変えても動作することから、図 10 のプログラムには汎用性があると言える。

```
% 物体 O1 を場所 P0 に移動する。
m([open,P0,O1,P1,O2,P2], [open,P0,O1,P0,O2,P0], g([open,P0,O1,P0,O2,P2])).
% 物体 O2 を場所 P0 に移動する。
m([open,P0,O1,P1,O2,P2], [open,P0,O1,P0,O2,P0], g([open,P0,O1,P1,O2,P0])).
```

図 12: タスク 2（物体 O1 と O2 を場所 P0 に移動するタスク）を解くプログラム。サブルーチンを 2 つ呼び出している。（そのうち 1 つの定義は図 13。）

```
%   1. 手を P1 に移動する。
m([open,P0,O1,P1,O2,P2], [open,P0,O1,P0,O2,P2], g([open,P1,O1,P1,O2,P2])).
%   2. O1 をつかむ。
m([open,P1,O1,P1,O2,P2], [open,P0,O1,P0,O2,P2], g([hld1,P1,O1,P1,O2,P2])).
%   3. 手を P0 に移動する。
m([hld1,P1,O1,P1,O2,P2], [open,P0,O1,P0,O2,P2], g([hld1,P0,O1,P0,O2,P2])).
%   4. 手をはなす。
% 定義不要。
```

図 13: 物体 O1 を場所 P0 に移動するプログラム。これはタスク 2 を解くプログラムから呼び出されるサブルーチンの 1 つである。物体 O2 を場所 P0 に移動するプログラムも同様に定義される。

```
?- episode([open,0,r,1,c,2], [open,0,r,0,c,0],Log),
 printList(Log).

[[open,0,r,1,c,2],[open,0,r,0,c,0],g([open,0,r,0,c,2])]
[[open,0,r,1,c,2],[open,0,r,0,c,2],g([open,1,r,1,c,2])]
[[open,0,r,1,c,2],[open,1,r,1,c,2],right]
[[open,1,r,1,c,2],[open,1,r,1,c,2],ret]
[[open,1,r,1,c,2],[open,0,r,0,c,2],g([hld1,1,r,1,c,2])]
[[open,1,r,1,c,2],[hld1,1,r,1,c,2],hold1]
[[hld1,1,r,1,c,2],[hld1,1,r,1,c,2],ret]
[[hld1,1,r,1,c,2],[open,0,r,0,c,2],g([hld1,0,r,0,c,2])]
[[hld1,1,r,1,c,2],[hld1,0,r,0,c,2],left]
[[hld1,0,r,0,c,2],[hld1,0,r,0,c,2],ret]
[[hld1,0,r,0,c,2],[open,0,r,0,c,2],release]
[[open,0,r,0,c,2],[open,0,r,0,c,2],ret]
[[open,0,r,0,c,2],[open,0,r,0,c,0],g([open,2,r,0,c,2])]
[[open,0,r,0,c,2],[open,2,r,0,c,2],right]
[[open,1,r,0,c,2],[open,2,r,0,c,2],right]
[[open,2,r,0,c,2],[open,2,r,0,c,2],ret]
[[open,2,r,0,c,2],[open,0,r,0,c,0],g([hld2,2,r,0,c,2])]
[[open,2,r,0,c,2],[hld2,2,r,0,c,2],hold2]
[[hld2,2,r,0,c,2],[hld2,2,r,0,c,2],ret]
[[hld2,2,r,0,c,2],[open,0,r,0,c,0],g([hld2,0,r,0,c,0])]
[[hld2,2,r,0,c,2],[hld2,0,r,0,c,0],left]
[[hld2,1,r,0,c,1],[hld2,0,r,0,c,0],left]
[[hld2,0,r,0,c,0],[hld2,0,r,0,c,0],ret]
[[hld2,0,r,0,c,0],[open,0,r,0,c,0],release]
[[open,0,r,0,c,0],[open,0,r,0,c,0],ret]
```

図 14: タスク 2 の実行ログの例。この例では物体 1 を動かしてから 2 を動かしているが、 2 を先に動かす別解もあり、実際にそのような解も出力される。

のを表現できるのではないかと考えている。このような表現にすることで、試行錯誤により解を探索する際に無意味な行動の選択をなくし、探索効率を上げることができるだろう。

現時点でのプログラミング言語としての RGoal に足りない機能として、ワーキングメモリへの読み書き、インデックスレジスタ、高階関数（関数ポインタ）、メタプログラミングなどがあり、アーキテクチャを少しずつ拡張することで、これらの機能に相当するものを実現していくことを計画している。また、ヒトの脳が解くべきタスクにおいて、これらの機能がどのように使われるかを今後具体的に分析する。

サブルーチンは手続きを抽象化するが、プログラミング言語にはデータ構造を抽象化する機構も必要である。これは前頭前野ではなく、感覚連合野が行う仕事であると考えている。具体的に脳がどのようなデータ構造を用いて外界を抽象化するのかを明らかにすることは、汎用人工知能構築に向けた非常に重要な課題であり、神経科学的知見や認知科学的知見を参考にしつつ検討を始めている。

### 5.7 スタックの必要性

サブルーチン呼び出しがネストする時、スタックがないとサブルーチンの再利用性が悪くなる理由を説明する。例えば、 $s \to m1 \to g$ という状態遷移を実現するプログラムと、そこからサブルーチンとして呼び出される $s \to m11 \to m1$ という状態遷移を実現するプログラムがあるとする。もしスタックがないと、状態 $m1$ に到達した時点で呼び出し元のサブゴール $g$ の情報がないので動作を継続できない。グローバルゴール $G$ を記憶しているとしても、

```
m(m1,G,g(g)).
```

というルールが存在してはじめて、 $m1 \to g$ という動作を継続することができる。スタックのないバージョンの RGoal [14] では、実際にそのような動作を継続するルールが学習により獲得されることにより動作していた。しかし、この方法ではグローバルゴールの数とサブルーチンの数の積だけ継続ルールを獲得する必要があり、それができないうちはサブルーチンの再利用ができなくなり、階層型強化学習の利点の 1 つが損なわれる。したがって、スタックはあった方が望ましいという結論に至った。

一方でスタックの導入は、サブルーチンの早期終了（サブゴールに到達する前に、別のサブゴールに切り替えること）の機会を奪い、行動の柔軟性を損なってしまうという欠点を持つ。例えばまわりの状況が変化して、現在のサブゴールを目指すことが無意味になることもあるだろう。必要に応じて現在のスタックの内容を破棄し、あらためてその時点での最適なサブゴールを設定することを可能にする機構を別途導入する必要があるだろう。

## 6 関連研究

DNC(Differentiable Neural Computers)[8] は、ニューラルネットワークで表現されたプログラムを、教師あり学習または強化学習により獲得するシステムである。外部メモリへの読み書きの機能を持っており、複雑なタスクを学習できるようにするためにメモリの構造に工夫がされている。

MagicHaskeller [6] はプログラムの入出力例から Haskell のプログラムを効率的に生成するシステムである。型的な制約の活用と、意味的に等価な部分式を検出し重複をなくすことで、出力プログラムの候補の数を小さくし、合成の効率を上げている。

## 7 まとめと今後

RGoal アーキテクチャが学習するテーブル $M(s, g, a)$ を、単一化の機構を使って大幅に圧縮する手法を提案し、いくつかのタスクを解くルールを記述する記述実験により、その有効性の予備的検討を行った。また、テーブル $M(s, g, a)$ を強化学習で合成されるプログラムと

見なし、そのプログラムを汎用性・再利用性の観点から考察することで、スタックの必要性など、いくつかの知見を得た。プログラムの汎用性・再利用性が高いということは、強化学習エージェントが獲得した知識の汎化能力が高いこと、知識獲得に必要なサンプルが少なくて済むことを意味する。提案手法は、汎用人工知能の性能を高めるための重要な技術の 1 つであると考える。

## 謝辞

本研究は JSPS 科研費 JP18K11488 の助成を受けたものです。

## 参考文献

[1] Kahneman, D., Treisman, A., and Gibbs, B. J., “The reviewing of object files: Object specific integration of information,” Cognit. Psychol., vol.24, pp.175-219, 1992.

[2] Thomas G. Dietterich, Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition, Journal of Artificial Intelligence Research 13, 227-303, 2000.

[3] M. Steedman, The Syntactic Process. The MIT Press, 2000.

[4] Hutter, M., A theory of universal artificial intelligence based on algorithmic complexity, Technical Report: cs.AI/0004001, 2000.
http://arxiv.org/abs/cs.AI/0004001

[5] 奥田次郎, 意図とその遅延後の実現：Prospective Memory の脳内過程 (<特集>意図研究のスペクトル), 人工知能学会誌/Journal of Japanese Society for Artificial Intelligence,20(4),418-424 (2005-07-01) , KJ00003364543.

[6] S. Katayama, Efficient Exhaustive Generation of Functional Programs using Monte-Carlo Search with Iterative Deepening , PRICAI 2008: Trends in Artificial Intelligence, Proceedings of 10th Pacific Rim International Conference on Artificial Intelligence, LNAI 5351, Springer Verlag, 199-211, 2008.

[7] Yuuji Ichisugi, A Computational Model of Motor Areas Based on Bayesian Networks and Most Probable Explanations, In Proc. of The International Conference on Artificial Neural Networks (ICANN 2012), Part I, LNCS 7552, pp.726–733, 2012.

[8] A. Graves, G. Wayne et al., Hybrid computing using a neural network with dynamic external memory, Nature 538, 471476, 2016.

[9] 一杉裕志, 疑似ベイジアンネットを用いた認知モデルのプロトタイピング手法の提案, 第 4 回 人工知能学会 汎用人工知能研究会 (SIG-AGI), 2016.

[10] Bacon, P.-L., Harb, J., Precup, D. The option-critic architecture. Proceedings of AAAI, 17261734, 2017.

[11] Naoto Takahashi and Yuuji Ichisugi, Restricted Quasi Bayesian Networks as a Prototyping Tool for Computational Models of Individual Cortical Areas, In Proc. of Machine Learning Research (AMBN 2017), Proceedings of Machine Learning Research, Vol.73, pp.188–199, 2017.

[12] Susumu Katayama, Computable Variants of AIXI which are More Powerful than AIXItl, 2018.
https://arxiv.org/abs/1805.08592

[13] 一杉裕志、高橋直人「脳における文の意味解析機構のモデル」第 8 回 人工知能学会 汎用人工知能研究会 (SIG-AGI), 2018.

[14] 一杉裕志, 高橋直人, 中田秀基, 佐野崇, RGoal Architecture:再帰的にサブゴールを設定できる階層型強化学習アーキテクチャ, 第 9 回 人工知能学会 汎用人工知能研究会 (SIG-AGI), 2018.