人工知能学会研究会資料
SIG-AGI-012-04

階層型強化学習 RGoal を用いた記号推論の実現手法の検討

Preliminary study of theorem proving with RGoal hierarchical reinforcement learning architecture

一杉裕志 [1*]　中田秀基 [1]　高橋直人 [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 aim at building a model of the information processing mechanism of the prefrontal cortex in the brain. For that purpose, we proposed an architecture of hierarchical reinforcement learning with unlimited recursive subroutine calls, RGoal. In this paper, we show that the slightly extended RGoal can execute a kind of symbolic inference, theorem proving. Moreover, we consider a mechanism of acquiring symbolic knowledge from agent's experience in the environment. These mechanisms are candidates of the models of symbolic inference and knowledge acquisition of human brain.

概要

我々はヒト前頭前野周辺の計算論的モデルの構築を目指しており、それに向けて再帰的にサブルーチン呼び出しが可能な階層型強化学習アーキテクチャ RGoal を提案している。本論文では RGoal にわずかな拡張を加えることで、記号推論の一例である定理証明ができることを示す。また、知的エージェントが経験により環境から記号的知識を獲得する機構について考察する。これらの機構は、ヒトの脳における記号推論と記号的知識獲得の機構の候補である。

1　はじめに

ヒトは不器用ながらも論理的な推論を行うことができるが、それはどのような機構によるものだろうか。例えば幼児であっても、次のような簡単な推論が行えるであろう。

「何か食べるものがないかな。きのうスナックとチョコレートがあったなあ。おにいちゃんはチョコレートは食べてないと言っていた。ということは、まだチョコレートが残っているから、食べられるはず。」

この推論は、抽象的な知識を組み合わせて、環境の隠れた状態を推定している。同様の推論を行う記号ＡＩには、現在のところ経験からの自動的な知識獲得の方法が確立されていないという問題がある。しかし一方で、記号ＡＩが扱う知識には、ニューラルネットワークが獲得するようなブラックボックス的な知識と比較して、以下の利点があるように思われる。

- 知識の汎用性が高く、複数の知識を組み合わせることで多様な推論が行える。
- 知識の抽象度が高く、汎化能力が高い。
- 知識のモジュラリティが高く、環境が部分的に変化したときにそれに合わせた変更がしやすい。
- 論理式で表現された知識は自然言語との相互変換が容易なため、他者との情報交換が容易になる。

ヒトの脳は、このような利点を持った記号ＡＩ的推論と、統計的手法とを巧妙に組み合わせているのではないだろうか。

我々はヒト前頭前野周辺の計算論的モデルの構築を目指しており、それに向けて再帰的にサブルーチン呼び出しが可能な階層型強化学習アーキテクチャ RGoal を提案している [3][4][6]。本論文では RGoal にわずかな拡張を加えることで、記号推論の一例である定理証明ができることを示す。ヒトが持つ記号推論の機構は

*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

1つとは限らないが、本稿で提案する実現手法は、その中でも比較的シンプルに実現可能なものの1つであろうと考えている。

本研究の動機の1つとして、記号AIにおける知識獲得の問題に対し、解決の糸口を見つけることがある。強化学習は知的エージェントが経験により環境から知識を得る機構である。記号推論を強化学習の枠組みの中で実装してみることで、記号的知識も環境から獲得できる可能性がある。また、RGoal アーキテクチャが前頭前野周辺の情報処理のよいモデルであるならば、その上で最も自然に動作する記号推論の機構を実装してみることで、脳内での記号的知識の表現方法や環境からの獲得方法についてのヒントが得られるはずである。

なお、本稿では行動ルールの価値の学習については述べないが、記号推論が強化学習の枠組みの中で実現されているため、経験を積むにつれて推論の効率が増していくことが期待される。この性質もヒトの推論機構の重要な特徴の1つである。

本論文は以下のような構成になっている。まず2章と3章で提案手法が前提とするRGoal アーキテクチャとテーブル圧縮手法について説明する。その後4章で提案手法を説明し、5章で具体的な推論タスクとその実行例について述べる。6章では記号的知識を環境から得る方法について考察する。7章で関連研究について述べ、最後に8章でまとめを行う。

2 RGoal アーキテクチャ

RGoal は再帰的なサブルーチン呼び出しが可能な階層型強化学習アーキテクチャである（図1）。RGoal では、サブルーチンは「任意の状態からある1つの状態（サブゴール）に向かう方策」と定義される。エージェントは過去の経験からの学習結果を用いて、いわばみずからの意志で目的（サブゴール）を設定し、その目的に向けて行動する。

RGoal の理論的基盤は明確かつシンプルなため、拡張がしやすいという利点がある。

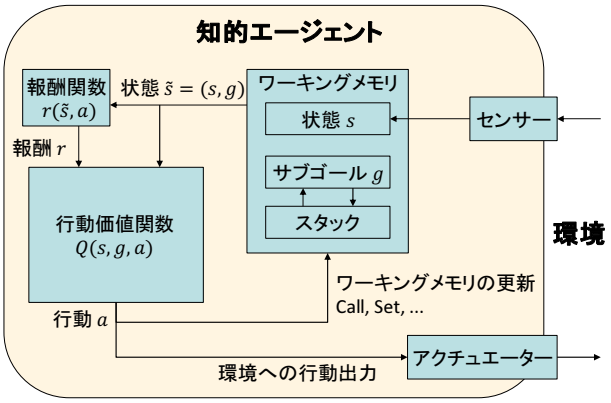

図 1: RGoal アーキテクチャ。通常の強化学習アーキテクチャとは異なり、ワーキングメモリとして状態 s、サブゴール g、スタックを持つ。行動価値関数 $Q(s,g,a)$ はワーキングメモリの一部 $\tilde{s}=(s,g)$ を参照する。行動 a は環境への運動出力またはワーキングメモリの状態の更新を行う。ただし、本稿の定理証明器の実装では環境とのインタラクションは行わず、ワーキングメモリの参照・更新のみが行われる。ワーキングメモリをエージェントから見た環境の一部と見なすことも可能で、その場合は通常の強化学習と本質的に変わらない。

3 パターンを用いたテーブル圧縮

本稿で述べる手法は、行動価値関数が [4] で述べた方法で圧縮表現されることを前提とする。この方法ではテーブルをパターンと値のペアからなる**ルール**の集合で表現し、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、テーブルサイズを大幅に減らすことで、汎化性能を向上させることを意図している（図2）。

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

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。左はサイズ 5x5 = 25 のテーブル。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。

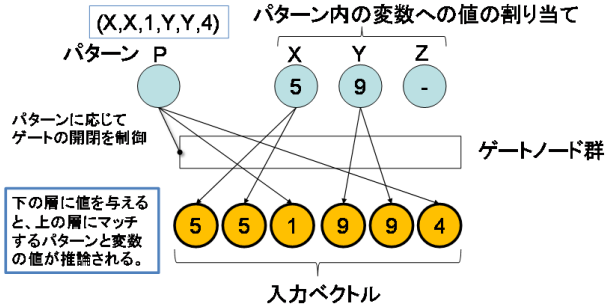

図 3: ベイジアンネットによるパターンマッチを行う回路の 1 つの案。(詳細は [4] を参照。) 左上のノード P はパターンの ID を値として持つ。ノード P は ID に対応するパターンに応じて、上の層と下の層の間の結合をゲートノードを使って適宜制御し、同じ値になるべきノードどうしを結合する。下の層のみに観測値を与えて事後確率最大の値の組を推論すると、入力にマッチするパターンと変数割り当てが上の層の値として表現される。

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。マッチする「最も特殊なパターン」が複数ある場合は、非決定論的にどれか 1 つのルールを選択する。

このようなテーブル圧縮とテーブル検索をベイジアンネットを用いて実現する 1 つの案を図 3 に示す。

4 提案手法

4.1 設計方針

本稿では Prolog が行うような後ろ向き連鎖による定理証明を、強化学習の枠組みの中で実現する方法を提案する。今回は、真であると仮定する命題が、あらかじめ与えられるものとする。(経験から獲得する方法については 6 章で考察する。) 後ろ向き連鎖による証明は、ゴールとなる命題の証明に必要な命題をサブゴールとして設定し、サブゴールを再帰的に証明していく。命題の再帰的証明に、 RGoal が持つ再帰的なサブルーチン呼び出しの機構を用いる。

4.2 命題のベクトル表現

本手法では、扱う命題を木構造で表現した時の深さと幅は有限であると仮定し、命題を固定長のベクトルで表現する。

我々の長期的目標はヒトと同じ仕組みで考える機械を作ることである。ヒトの脳は固定した神経回路で知識の獲得と推論の両方を実現しており、それに適したデータ表現はおそらく固定長のベクトルであろう。我々はヒトが脳内で扱いきれないような巨大な論理式を扱うことは目標とはしていない。

今回のプロトタイプ実装では 以下の形の論理式で表現できる命題のみを扱うものとする。

- 述語の引数は高々 2 つとする。例： P(a1,a2)
- 論理式は A 、 A → B 、 A ∧ B → C のいずれかの形とする。ただし A, B, C は述語である。

以上の仮定のもとで、すべての述語を長さ 3 のベクトル、すべての論理式を 3 つの述語（前提 1 、前提 2 、結論）をつなげた長さ 9 のベクトルで表現する。使用しないベクトルの要素には記号 “O” を入れる。例：

```
P(a,b) ∧ Q(c) → R : (P a b Q c O R O O)
P(a,b) → Q(c)     : (P a b O O O Q c O)
P(a,b)            : (O O O O O O P a b)
```

4.3 状態

エージェントにとっての状態 s とサブゴール g の値はどちらも、前提 1 、前提 2 、結論から構成される命題（長さ 9 のベクトル）であるとする。状態 s は CPU のレジスタのような役割を果たし、スタックと合わせて証明の進行状況を表現する。

状態 s には、注目する命題以外にも様々な文脈情報を含ませてもよいが、今回は実装を極力シンプルにするため、必要最低限の情報のみとした。

4.4 行動

エージェントが取り得る行動 a の値は、Call(m), Set(m), Fail のいずれかとする。今回の実装では環境に対する行動出力はない。各行動は以下のような動作をする。

- Call(m) ：サブルーチン呼び出し
  現在のサブゴール g をスタックに積んで、 m を新しいサブゴールとする。
- Set(m) ：状態の書き換え
  現在の状態 s を m に置き換える。
- Fail : 状態のリセット
  スタックを破棄し、 s, g, を初期値にリセットする。

命題 A を表現するルール:
Q((_, _ ,_), (O, O, A), Set(O, O, A)).

命題 A → B を表現するルール:
Q((_,_,_), (O,O,B), Call(O,O,A)).
Q((O,O,A), (O,O,B), Set(O,O,B)).

命題 A ∧ B → C を表現するルール:
Q((_,_,_), (O,O,C), Call(A,O,O)).
Q((A,O,O), (O,O,C), Call(A,B,O)).
Q((A,B,O), (O,O,C), Set(O,O,C)).
Q((_,_,_), (A,O,O), Call(O,O,A)).
Q((O,O,A), (A,O,O), Set(A,O,O)).
Q((A,O,O), (A,B,O), Call(O,O,B)).
Q((O,O,B), (A,B,O), Set(A,B,O)).

図 4: RGoal において定理証明を実行する行動ルールの記述方法。本来は学習によってこれらのルールが獲得されるはずであるが、今回のプロトタイプ実装では行動ルールは手で記述して与えている。ここで、 “O” は空、 “_” はワイルドカードを表す。この図では各述語を 1 つの記号で表しているが、実際には 4.2 節で述べたように各述語は長さ 3 のベクトルで表現される。

証明を進めていき選択できるルールがなくなった場合は Fail が選択されるものとする。Prolog では選択可能なホーン節がなくなったら証明の途中までバックトラックするが、本手法では途中ではなく最初からやり直す。

4.5 行動価値関数のテーブル

本手法では行動ルールは Q(s,g,a) の形をしたパターンで記述する。ルールの書き方を図 4 に示す。この書き方に従って具体的なルールを記述した例が図 8 である。

この書き方で書かれたルールの意味は、以下のように解釈できる。

- 命題 A が真であると仮定されているとき：
  命題Aの証明が目的の場合、前提によらず、無条件に証明されたことにしてよい。
- 命題 A → B が真であると仮定されているとき：
  命題Bの証明が目的の場合、まず命題Aを証明する。Aが証明ずみならBは証明されたことにしてよい。
- 命題 A ∧ B → C が真であると仮定されているとき：
  命題Cの証明が目的の場合、命題A、Bを証明してその結果を前提と見なす。A，Bが前提にあればCは証明されたことにしてよい。

4.6 行動選択の方法

行動選択は以下のように行う。まず、現在の状態とサブゴールの値にパターンマッチ可能なパターン s, g を持つ行動ルール $Q(s, g, a)$ を 1 つ選択する。その時の変数の値の割り当てをパターン a に適用した結果を、選択された行動とする。例えば現在の状態が (a,b) 、サブゴールが (c,d) で、それらとパターンマッチ可能な行動ルール Q((a,X),(Y,Z),Set(X,Y)) が選択されたならば、変数の値は X=b, Y=c, Z=d となるため、選択された行動は Set(b,c) となる。

選択可能な「最も特殊なパターン」を持つ行動ルールが複数ある場合、今回の実装では softmax 関数で得られる確率分布を用いて、価値の高いルールほど高い確率で選択するようにしている。今回はすべての行動ルールの価値を 0 で初期化しているので、価値の学習を行わない場合は、一様分布に従って 1 つを選択することになる。

なお、証明が一定時間内に成功しなかった場合は強制的に Fail が実行されるものとする。

4.7 変数束縛についての制限

Prolog は未束縛の変数をデータ構造の中に埋め込んで再帰呼び出しの引数として渡すことができる。しかし、このような強力すぎる機構が脳にあるとは考えにくい。また、そのような機構はソフトウエアで実装する際にも手間がかかる。そこで今回は、変数の共有構造は推論 1 ステップの中でのみ有効であるような実装とした。

具体的には現在の実装は以下のようになっている。状態 s とサブゴール g とのパターンマッチの結果、未束縛の変数がパターン a の中に残ったまま選択された場合、その未束縛変数の値は特殊な値 PHI とする。これにより、推論の次のステップに進むときに未束縛変数のアイデンティティは失われ、 PHI という値だけが伝わる。値 PHI はパターンマッチの際に、任意の値とマッチするものとする。これにより、 PHI は未束縛変数と似た振る舞いをする。このような実装でも、5 章で示すように、簡単な推論は実行することができる。

なお、変数束縛に何らかの制限を入れるとしても、このやり方が唯一のやり方とは限らない。制限を入れれば推論機構の能力も制限され、場合によっては健全性（証明された命題は必ず真であるという性質）も失われてしまう。例えば現在の実装ではサブゴールに PHI が 2 つ以上現れる場合、もともとの変数のパターンを無視してパターンマッチが成功する場合があるので、健全性が損なわれる。（しかし 6 章で述べるように健全性を損なう行動ルールが学習により選択されなくなる機構があるならば、長期的には問題は起きなくなるだろ

```
1: procedure EPISODE(s, g)
2:     stack ← empty
3:     Choose a rule which matches (s, g) and get a from it.
4:     while not (stack is empty and a = Return) do
5:         # Take action.
6:         if a is Return then
7:             s' ← s; g' = stack.pop()
8:         else if a is Call(m) then
9:             stack.push(g)
10:            s' ← s; g' ← m
11:        else if a is Set(m) then
12:            s' ← m; g' ← g
13:        else if a is Fail then
14:            stack ← empty
15:            s', g' ← initial values
16:        else
17:            Error
18:        # Choose action.
19:        if s' matches g' then
20:            a' ← Return
21:        else
22:            Choose a rule which matches (s', g') and get a' from it.
23:        s ← s'; g ← g'; a ← a'
    return
```

図 5: 今回実装した証明アルゴリズムの疑似コード。スタックのある RGoal のアルゴリズム [4][6] に新たに Set, Fail というアクションを追加した。EPISODE(s, g) は任意の初期状態 s からゴール g に到達するまで行動を繰り返す。ここには価値を学習するコードは含まれていないが、 Sarsa で学習するように修正することは容易である。

う。）一方で、適切な制限はサブルーチンのモジュラリティを増し、知識獲得を容易にするという利点を生む可能性がある。どのような制限を入れるのが工学的に妥当なのか、また、実際にヒトの脳はどのような制限を入れているのかについて検討していくことが今後の課題である。

4.8 アルゴリズムの疑似コード

本章で述べた定理証明を行うアルゴリズムの疑似コードを図 5 に示す。既存の RGoal のアルゴリズムに Set, Fail を追加するというわずかな拡張を施したものになっている。アルゴリズムは依然としてシンプルであり、脳

```
1: 昨日はスナックがあった。
2: 昨日はチョコレートがあった。
3: 兄はチョコレートを食べていない。
4: 昨日Xがあって、兄がXを食べていないなら、Xは今日もまだある。
5: Xが今日あるなら、Xを食べられる。

何か食べられるXはあるか？
```

図 6: 例題タスク。いくつかの知識を組み合わせて、環境の隠れた状態を推定する。

```
1: exist(yesterday, snack).
2: exist(yesterday, chocolate).
3: notEat(brother, chocolate).
4: exist(today,X) :- exist(yesterday,X), notEat(brother,X).
5: canEat(X) :- exist(today, X).

実行結果：

?- canEat(X).
X = chocolate.
```

図 7: 例題タスクの Prolog による記述と実行結果。定理証明の結果の副産物として変数 X の値も得られている。

内にある神経回路で十分に実現可能であるように思われる。

5 ルール記述例と実行例

今回は、幼児が行うような簡単な推論を例題タスクとして用いる（図 6）。

この推論を行うプログラムの Prolog による記述を図 7 に示す。 Prolog のプログラムはホーン節（ 0 個以上の前提から 1 つの結論を導く形をした命題）の集合として記述される。図 7 の 4 行目のホーン節の 1 つ目のサブゴール exist(yesterday, X) が実行されると、選択可能なホーン節は 1 行目または 2 行目となる。Prolog では選択可能な最初のホーン節をまず選択して証明を進めて、もし証明が失敗したら最初のホーン節の選択をなかったことにし（バックトラック）、あらためて 2 つ目のホーン節を選択してやり直す、という動作になる。

この Prolog の記述と同じ内容を、提案手法を用いて記述したものが図 8 である。また、図 9 は、証明過程でエージェントが選択した行動を並べた実行ログである。提案手法では、選択可能な行が複数ある場合は softmax 関数を用いて非決定論的に選択し、失敗したら（途中までのバックトラックではなく）最初まで戻って証明をやり直す。非決定論的な選択が行われる個所

```
Q((__,__,__,__,__,__,__,__,__), (__,__,__,__,__,__,__,__,__), Fail)

Q((__,__,__,__,__,__,__,__,__), (O,O,O,O,O,O,Exist,Yesterday,Snack), Set(O,O,O,O,O,O,Exist,Yesterday,Snack))

Q((__,__,__,__,__,__,__,__,__), (O,O,O,O,O,O,Exist,Yesterday,Chocolate), Set(O,O,O,O,O,O,Exist,Yesterday,Chocolate))

Q((__,__,__,__,__,__,__,__,__), (O,O,O,O,O,O,NotEat,Brother,Chocolate), Set(O,O,O,O,O,O,NotEat,Brother,Chocolate))

Q((__,__,__,__,__,__,__,__,__), (O,O,O,O,O,O,Exist,Today,x), Call(Exist,Yesterday,x,O,O,O,O,O,O))
Q((Exist,Yesterday,x,O,O,O,O,O,O), (O,O,O,O,O,O,Exist,Today,x), Call(Exist,Yesterday,x,NotEat,Brother,x,O,O,O))
Q((Exist,Yesterday,x,NotEat,Brother,x,O,O,O), (O,O,O,O,O,O,Exist,Today,x), Set(O,O,O,O,O,O,Exist,Today,x))
Q((__,__,__,__,__,__,__,__,__), (Exist,Yesterday,x,O,O,O,O,O,O), Call(O,O,O,O,O,O,Exist,Yesterday,x))
Q((O,O,O,O,O,O,Exist,Yesterday,x), (Exist,Yesterday,x,O,O,O,O,O,O), Set(Exist,Yesterday,x,O,O,O,O,O,O))
Q((Exist,Yesterday,x,O,O,O,O,O,O), (Exist,Yesterday,x,NotEat,Brother,x,O,O,O), Call(O,O,O,O,O,O,NotEat,Brother,x))
Q((O,O,O,O,O,O,NotEat,Brother,x), (Exist,Yesterday,x,NotEat,Brother,x,O,O,O), Set(Exist,Yesterday,x,NotEat,Brother,x,O,O,O))

Q((__,__,__,__,__,__,__,__,__), (O,O,O,O,O,O,CanEat,x,O), Call(O,O,O,O,O,O,Exist,Today,x))
Q((O,O,O,O,O,O,Exist,Today,x), (O,O,O,O,O,O,CanEat,x,O), Set(O,O,O,O,O,O,CanEat,x,O))
```

図 8: RGoal において例題タスクを実行する行動ルールを Q(s,g,a) の形のパターンで記述したもの。ここでは Java のコンベンションに従い、定数は大文字、変数は小文字で始まる記号で表している。先頭のルールは、他に選択すべきルールがないときに選択され、命令 Fail を実行する。

```
Call(O,O,O,O,O,O,CanEat,PHI,O)
        Call(O,O,O,O,O,O,Exist,Today,PHI)
                Call(Exist,Yesterday,PHI,O,O,O,O,O,O)
                        Call(O,O,O,O,O,O,Exist,Yesterday,PHI)
                                Set(O,O,O,O,O,O,Exist,Yesterday,Snack)    *1
                                Return
                        Set(Exist,Yesterday,Snack,O,O,O,O,O,O)
                        Return
                Call(Exist,Yesterday,Snack,NotEat,Brother,Snack,O,O,O)
                        Call(O,O,O,O,O,O,NotEat,Brother,Snack)
                                Fail
Call(O,O,O,O,O,O,CanEat,PHI,O)
        Call(O,O,O,O,O,O,Exist,Today,PHI)
                Call(Exist,Yesterday,PHI,O,O,O,O,O,O)
                        Call(O,O,O,O,O,O,Exist,Yesterday,PHI)
                                Set(O,O,O,O,O,O,Exist,Yesterday,Chocolate)    *2
                                Return
                        Set(Exist,Yesterday,Chocolate,O,O,O,O,O,O)
                        Return
                Call(Exist,Yesterday,Chocolate,NotEat,Brother,Chocolate,O,O,O)
                        Call(O,O,O,O,O,O,NotEat,Brother,Chocolate)
                                Set(O,O,O,O,O,O,NotEat,Brother,Chocolate)
                                Return
                        Set(Exist,Yesterday,Chocolate,NotEat,Brother,Chocolate,O,O,O)
                        Return
                Set(O,O,O,O,O,O,Exist,Today,Chocolate)
                Return
        Set(O,O,O,O,O,O,CanEat,Chocolate,O)    *3
        Return
```

図 9: 実行ログ。エージェントが実行したアクションを、サブルーチン呼び出し階層の深さに応じてインデントを付けて示したもの。*1 と *2 は非決定論的に選択されたルールであり、最初の選択 *1 はのちに Fail を引き起こしている。 2 度目の選択 *2 では証明に成功し、証明の最後 *3 で変数 x の適切な値 (Chocolate) も得られている。

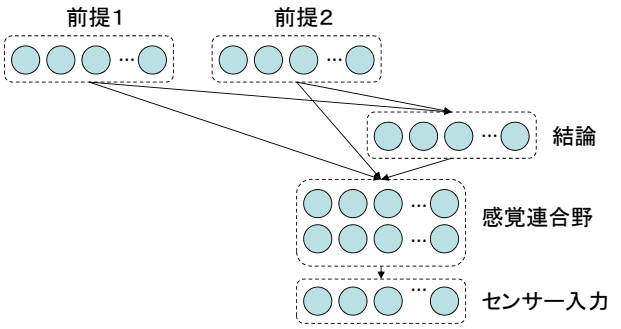

図 10: 命題の真偽値を環境にグラウンディングさせるためのベイジアンネットの 1 つの案。2 つの命題（前提 1 と前提 2）からセンサー入力のノードの値を生成する生成モデルの形をしている。

では最初は exist(yesterday,snack) が選択され（図 9 の *1)、その後選択可能なルールがなくなり Fail が選択されている。2 度目は exist(yesterday,chocolate) が選択され（図 9 の *2)、証明が成功している。

この実行例では、ゴールとなる命題 canEat(X) の証明に成功した上、変数 X の値が chocolate であると正しく推論されている（図 9 の *3)。(ただし、一般には証明が成功しても引数の値まで得られるとは限らない。)

6 記号的知識の獲得方法についての考察

本稿では真と仮定する命題は行動ルールの形で最初から与えたが、エージェントが経験から真とみなせる命題を獲得できるようにするためにはどのようにすればよいだろうか。

人間が日常的に扱う命題は、環境の状態を抽象化して表現したものであろう。例えば exist(today, chocolate) という命題は、「今家にチョコレートがあるという状態」を抽象化しており、環境の状態に対応して真偽値が決まる。

原理的には、ある種のオートエンコーダーで環境の状態を教師なし学習することで、真と見なせる命題が獲得できるはずである。例えば図 10 のような構成のベイジアンネットを用いてそれが実現できるかもしれない。このネットワークは命題からセンサー入力のノードの値を生成する生成モデルの形をしており、環境からのセンサー入力を再下端のノードの値として入れてパラメタを学習することで、環境の状態を抽象化した表現が最上位層に獲得される。(学習を成功させるためには非常に強い事前知識の作り込みや、適切な学習カリキュラムの設定が必要となるだろう。) 学習が成功すれば、命題の意味が環境の状態と結び付けられる。上位層で表現された命題が現在の環境において真であるか偽であるかは、ネットワーク全体の同時確率が 1 に近いか 0 に近いかに対応することになる。

一般に環境はセンサー入力に反映されない「隠れた状態」を持ちうる。例えば視野の外側の状態や、過去の状態などはセンサーで感知できない。その場合でも学習が不完全データからのベイジアンネットのパラメタ学習の問題に帰着されることには変わりはなく、EMアルゴリズム等を用いることで原理的には学習可能である。

以上の機構がうまく動いたとしても、正しい推論を行う行動ルールをエージェントが獲得する仕組みは自明ではない。現在のところ以下に説明する機構を考えている。例えば命題Aが「ユニコーンが実在する」のような真でない命題であるにもかかわらず、次のような行動ルールがあるとする。

```
Q((_,_,_),(O,O,A),Set(O,O,A))
```

(この行動ルールは図 4 で説明したように、命題Aが真であると仮定している。) もし命題Aが真になり得ないことを図 10 のネットワークが学習済みならば、この行動ルールが選択されても、ネットワーク全体の同時確率が 0 になるため、状態は (O,O,A) に変化しない。そうするとサブゴール (O,O,A) は永遠に達成されず、この行動ルールの価値（サブゴール達成コストの期待値）はマイナス無限大に近づいていく。するとこの行動ルールは決して選択されなくなるので、実質的に存在しないに等しくなっていく。同様にして、真でない命題を推論してしまう行動ルール（健全性を損なう行動ルール）は長期的には価値が下がっていく。

これらの機構の妥当性の検証を今後行っていく必要がある。

7 関連研究

RRL(Relational Reinforcement Learning)[1] は、強化学習と帰納論理プログラミングを組み合わせたシステムであり、タスクを試行錯誤によって繰り返し解きながら、行動価値関数を圧縮表現する論理決定木と呼ぶ構造を学習する。行動価値関数を記号表現を使って圧縮することで極めて強力な汎化能力を得ようとする点で、我々の研究と目標は同じである。我々が用いようとしているパターンマッチとサブルーチンを組み合わせた表現方法は、論理決定木よりは表現力が強いものになる。

DNC(Differentiable Neural Computers)[2] 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。

rlCoP[5] は強化学習を用いた定理証明システムである。定理証明タスクを繰り返してその経験から方策の事前確率と証明状態の価値を学習し、未知の問題の証明の効率を上げることを目標としている。アーキテクチャはコンピュータ囲碁に強く触発されており、証明の 1 ステップの選択をゲームの 1 手の選択と同様と考えて、行動選択にモンテカルロ木探索を用いている。我々の研究は、証明の効率の向上だけでなく、ヒトと同様の仕組みによる推論・知識獲得を実現することを目指しており、研究の目的が異なっている。

筆者らが提案した RGoal [3][6] は思考モードと呼ぶ、一種の演繹推論の機構を有しているが、本稿で述べた機構は思考モードとは別の演繹推論の機構である。RGoal の思考モードとは、学習した行動価値関数 Q(s,g,a) を環境のモデルとみなして脳内シミュレーションし、未経験の行動の価値を推定する機構である。一方、本稿で述べた機構は、教師なし学習で獲得した宣言的知識（これも環境のモデルの一部である）を組み合わせて、隠れた状態などを推定するための機構である。

8　まとめと今後

以前提案した RGoal アーキテクチャにワーキングメモリの書き換えの機能を追加し、それにより Prolog の実行に似た記号推論が行えることを示した。使用するデータ構造とアルゴリズムは十分にシンプルであり、おそらく脳内の神経回路でも実現可能である。記号推論は強化学習の枠組みの中で実現されているため、経験を積むにつれて推論の効率が増していくことが期待される。今回はプロトタイプ実装により簡単なタスクの動作確認をしたにとどまったが、今後提案手法をベースにして、他者の心の状態の推定や言語理解、合目的的な発話などのモデルの実現につなげていきたい。

本稿では後ろ向き連鎖による定理証明の 1 つの実現手法を述べたが、他にもヒトは前向き連鎖による推論の機構も持っている可能性がある。そのような機構のモデルの実現も今後検討していく。

提案手法の神経科学的妥当性・認知科学的妥当性に関する検討はまだ十分ではない。提案手法は 4.7 節で述べたように変数束縛に関して自明でない制限を設けており、この制限のせいで証明能力や変数の値の推論能力に制限が生じる。同様の制限がヒトの論理的推論にもあるかどうかは興味深い問題であり、おそらく何らかの認知科学的実験により検証可能であろう。

本稿ではまた、提案手法をベースとして、教師なし学習と強化学習を組み合わせて記号的知識を獲得していく方法について考察した。考察した手法は今後の実証が必要ではあるが、汎用人工知能の自律的な知識獲得の原理として有望であると考えている。

謝辞

本研究に関して議論をしていただいた竹内泉氏に感謝いたします。

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] Saso Dzeroski, Luc De Raedt, and Kurt Driessens, Relational reinforcement learning. Machine Learning, 43, pp.7–52, 2001.

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

[3] 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.

[4] 一杉裕志、高橋直人、中田秀基、佐野崇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, 第 10 回 人工知能学会 汎用人工知能研究会 (SIG-AGI), 2018.

[5] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, Mirek Olšák, Reinforcement Learning of Theorem Proving, In Proc. of 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), 2018.

[6] Yuuji Ichisugi, Naoto Takahashi, Hidemoto Nakada, Takashi Sano, Hierarchical Reinforcement Learning with Unlimited Recursive Subroutine Calls, In Proc. of 28th International Conference on Artificial Neural Networks (ICANN 2019), 2019. (to appear)