人工知能学会第二種研究会資料
汎用人工知能研究会
SIG-AGI-020-03

汎用人工知能のためのプログラム合成対象言語 Pro5Lang のエピソード記憶機構

Episodic Memory Mechanisms of Pro5Lang, a Target Language for Program Synthesis for Artificial General Intelligence

一杉裕志[1*]　中田秀基[1]　高橋直人[1]　竹内泉[1]　佐野崇[2]
Yuuji Ichisugi[1]　Hidemoto Nakada[1]　Naoto Takahashi[1]　Izumi Takeuti[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] 東洋大学 情報連携学部情報連携学科
[2] Faculty of Information Networking for Innovation And Design, Toyo University

Abstract: This paper describes an associative memory mechanism of Pro5Lang, a target language for program synthesis for artificial general intelligence. This language has the features of logic programming languages and machine languages. The associative memory stores only proven propositions that are obtained in the process of proof search. This proposed mechanism is also a hypothetical computational model that clearly explains one of the major roles of episodic memory in the brain. We preliminarily examined the expressive power of Pro5Lang and its suitability as a target language for synthesis by writing several test programs.

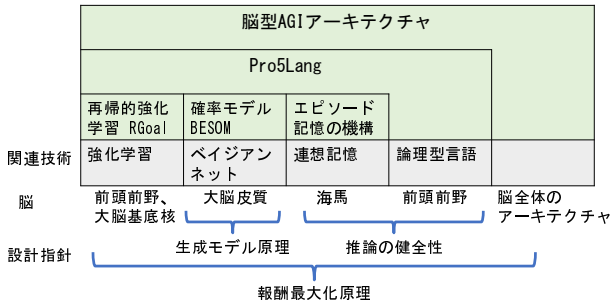

図 1: 設計を進めている脳型 AGI アーキテクチャの構成と、その中における Pro5Lang の位置づけ。

1　はじめに

十分な性能を持つプログラム合成システムは汎用人工知能 (Artificial General Intelligence, AGI) と見なせるだろう。プログラム合成の性能を上げるためには合成対象となるプログラミング言語を適切に設計する必要がある。我々はこれまで再帰的強化学習 RGoal [1] を用いたプログラム合成システムの実現に向けた研究を行ってきた [2][3][4][5][6][7]（図 1）が、これまで扱った合成対象言語は限られたサイズのワーキングメモリしか持っておらず、プログラムの表現力は著しく低かった。本稿では、表現力を大きく向上させるための連想記憶機構を提案する。この機構はヒトの**宣言的記憶（エピソード記憶と意味記憶）**の計算論的モデルの有望な候補でもある。

本稿では我々が以前から設計・実装を進めているプログラム合成対象言語を **Pro5Lang** (Probabilistic Proven-Proposition Processing Programming Language, 確率的証明済み命題処理プログラミング言語) と名付ける。この言語は**論理型言語**（数理論理学を基礎にしたプログラミング言語）と**機械語**（コンピューターを構成する論理回路が直接解釈実行できる言語）の特徴を合わせ持ったプログラミング言語である。Pro5Lang のプログラムの実行は証明探索と見なせる。情報の記憶場所としてレジスタと連想記憶装置を有し、そこに証明済みの命題のみを記憶する。

Pro5Lang の設計にあたっては、**証明の健全性** (正しい命題のみが証明されるという性質) が保たれることを設計指針の１つとして重視する。これによりアーキテクチャ全体の動作の見通しが劇的に良くなると同時に、細部の仕様の設計が容易になる。同時に、エピソード記憶・意味記憶に関する神経科学的・認知科学的知見も設計のヒントとする。こうして設計・実装した Pro5Lang のプロトタイプシステムの上で、いくつかのテストプログラムを記述することで、Pro5Lang のプログラム

*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

合成対象言語としての適性を予備的に検証した。

本稿は以下のような構成になっている。まず 2 節で解決すべき問題について述べる。3 節では Pro5Lang の概要と提案する宣言的記憶機構、4 節ではテストプログラムの例を説明する。5 節では正しい宣言的知識の選別方法の問題について議論し、6 節で関連研究について述べ、最後に 7 節でまとめを述べる。

2 解決すべき問題

2.1 メモリ機構の適切な設計

プログラミング言語が十分な表現力を持つためには大容量のメモリ機構が必須である。しかし通常のプログラミング言語はメモリアクセスの自由度が高いため、プログラム合成のための解の探索空間が広くなり、合成が難しくなる。そこで、合成しやすくかつ十分な表現力を持ったメモリ機構をどう設計すればよいかが問題となる。

また、システムの性能を上げるためには解くべきタスクに特化させる必要がある。ヒトが日常的に行うタスクには具体的に以下のものがある。

- **物体操作**： 視覚と手足を用いた、環境中の物体を操作する様々な作業。物体や自分自身の移動、道具の作成、狩猟・採集、調理など。
- **推論**： 過去の記憶や知識を用いた、現在・過去・未来における環境の隠れた状態の推定。「どこに何がある」「誰が何を知っている」といった推論や、「いつどこで何が起きる」という将来のイベントの予測など。
- **対話**: 聴覚と発声器官・調音器官を用いた、他者との意思疎通。事実の伝達、要求、命令、約束、謝罪、感謝など。

ヒトに似た知能の実現を目指すならば、これらのタスクに特化したメモリ機構を設計する必要がある。逆に、普通のプログラミング言語で簡単に書けるクイックソートの問題などは、ヒトは脳内で簡単に解ける必要はない。ヒトに必要のない問題が過剰に解きやすく設計されていると、その代わりにヒトが解くべき問題が解きにくくなる恐れがあり、その点に注意して合成対象言語を設計する必要がある。

2.2 エピソード記憶の計算論的役割の解明

ヒトのエピソード記憶は宣言的記憶の１つで、「いつどこで何をした」といった個人が体験した出来事に関する記憶である。脳のアーキテクチャをヒントに AGI アーキテクチャを作るならば、エピソード記憶の機構は不可欠であろう。しかし、エピソード記憶の脳全体の情報処理における役割を説明する計算論的モデルは現時点では存在せず、仕様の詳細を決める設計指針が得られていないという問題がある。海馬は、エピソード記憶に関連するパターン補完とパターン分離と呼ばれる機能を実現していると考えられておりその詳細な神経回路モデルの研究も進んでいる [8] が、海馬の機能と脳全体の情報処理との関係については未だ明らかではない。

実際、筆者がエピソード記憶の機構を実装しようと考えた際、その仕様に関して以下の点を含む数多くの疑問が生じた。

- エピソードを想起することは過去の大脳皮質の発火パターンの再現なのか？しかし大脳皮質全体の発火が再現されたらその瞬間、それまで行っていた情報処理コンテキストが失われてしまうのではないか？
- 想起のタイミングや連想に用いる類似度はどのような基準で決めるべきなのか？
- 海馬から皮質への記憶の固定化は具体的にどう実現されるのか？

これらの疑問を解決し、エピソード記憶の役割を明解に説明する計算論的モデルがもしあれば、脳の理解が深まるとともに、脳を模倣した AGI の実現も加速するはずである。

3 提案する宣言的記憶機構

3.1 証明に関する用語の定義

この節では本稿で用いる証明に関する用語を非形式的に簡単に定義する。これらの用語は数理論理学における用法にほぼ従っている。数理論理学とは数学の一分野であり、推論や証明の過程を定式化して研究の対象とする学問である。

命題とは真か偽かを問うことができる文である。**論理式**とは命題を式を使って形式的に表現したものである。以下の説明では命題と論理式を厳密に区別せずに使う。

論理体系は推論規則の集合である。**推論規則**は**前提**となる命題がすべて真であるときに、**結論**となる１個の命題が真であると推論するための規則である。「命題 P,Q が真のとき命題 R は真である」と推論する推論規則を P,Q ⊢ R と書く。推論規則を適用して推論された R を本稿では**証明済み命題** (proven-proposition) と呼ぶ。前提となる命題が１つもない推論規則も許され

る。例えば ⊢ P は、無条件に結論 P を真であると推論する推論規則である。(P はいわば公理である。)

3.2 「エピソードは証明済み命題である」

この節では、脳内における宣言的記憶の役割を計算論的に説明する１つの仮説について述べる。Pro5Lang の宣言的記憶機構はこの仮説に基づいて設計される。

論理体系が与えられた時、ある命題がその論理体系において成り立つかどうか機械的に証明できる場合がある。ここでは説明の理解の助けとするために、１つの極めてシンプルな証明アルゴリズムの疑似コードを図 2 に示す。このアルゴリズムでは、推論規則を適用するたびに証明済み命題の数が単調に増加する。そして、偶然に目的とする命題が証明されたときに手続きを終了する。

Pro5Lang の証明探索戦略 [3] は図 2 のような単純なアルゴリズムとは異なり、ゴールからさかのぼって必要な前提を再帰的に証明することを試みる。また、推論規則の選択は完全にランダムではなく、報酬最大化(証明コスト最小化)につながるものを高い確率で選択する。しかし、 Pro5Lang においても原則として証明済み命題の数が単調に増加する。(後述のように忘却も許される。)Pro5Lang では証明済み命題集合 s を宣言的知識の集合と見なしている。これが本稿が提示する宣言的記憶の役割の計算論的な説明である。

ヒトの脳がこの仮説に従っているならば、宣言的記憶は推論(証明)の前提に用いる目的で保持され、ヒトが何かを推論するたびに脳は推論された命題を宣言的記憶に追加していく、ということになる。

自分自身が体験したエピソードも証明済み命題と見なせる。例えば脳内のニューラルネットワークがセンサー入力を通じて環境の状態を認識した結果が何らかの数値ベクトルで表現されるとする。数値ベクトルが (0.2, 0.7, ...) という値の時、その値は F1(0.2) ∧ F2(0.7) ∧... という命題を表現していると解釈することができる。つまり脳が外界を認識するということは、外界の状態を表現する命題を、センサーの状態を証拠として証明することに等しいのである。

Pro5Lang は、間違った知識を持っている場合は真ではない命題を証明してしまう。しかし、理想的な条件下では Pro5Lang は正しい推論だけを行う。より厳密にいえば、プログラムが正しい推論をする推論規則だけで構成されている場合[1]は、証明される命題は必ず正しいという性質(証明の健全性)を持つ。ただし、正しい命題が必ず証明されるという性質(完全性)は持たない。

[1] [4] で述べた機構により、エージェントが経験にもとづく学習を繰り返すうちに、正しい推論をする推論規則だけが残るはずである。

3.3 宣言的記憶機構の概要

証明の健全性という数理論理学的な要請に反しない範囲で、神経科学的知見・認知科学的知見とできるだけ整合性を持つような宣言的記憶機構を設計した。その主要な特徴を以下に述べる。

レジスタおよび連想記憶装置に入れる値は証明済み命題のみに限定する。情報処理の最小単位(証明済み命題)が自己完結的に意味を持つため、記憶域管理(不要な知識の忘却など)が容易になる。また、合成過程にある不完全なプログラムの実行においてもデータ構造が破綻しにくいという利点がある。プログラムの実行順序の自由度も高くなるため、時間に余裕があるときに将来の行動計画を立てておくといったこともおそらく容易に実現可能と思われる。

証明した命題の証明済み命題集合への追加、すなわち連想記憶機構への情報の書き込みは、推論のたびに自動的に行われる。これによりプログラム側の負担を減らし、プログラム合成の性能を上げることを狙う。

宣言的知識の想起には、自動的な想起と能動的な想起の２種類を用意する予定である。(現在は能動的想起だけが実装されている。)

エピソードの想起はある瞬間の脳の内部状態のすべてを再現するのではなく、１つの証明済み命題のみを再現するものとする。これにより、複数の命題から別の命題を推論するといった情報処理が可能になる。

宣言的知識の想起の機構も証明の健全性を損なわないように設計される(3.7 節)。

宣言的知識は、証明済み命題(エピソード記憶)と**意味記憶**に分かれる。Pro5Lang における意味記憶は、複数の証明済み命題を圧縮・抽象化したもので、[5] で述べた方法で帰納推論することで獲得されるようにする予定である[2]。(ただし現実装では意味記憶も最初から手で与える。)過汎化して間違った意味記憶を獲得してしまう可能性があるため、間違った意味記憶を選別し忘却する仕組みを別途用意する必要がある。これについては 5 節で議論する。

記憶域は有限なので、正しい宣言的知識であっても、適宜忘却するものとする。図 2 のアルゴリズムや Pro5Lang においては、証明済み命題集合の要素を時々ランダムに選んで取り除いても、健全性は失われない。ただし、証明により時間がかかったり、証明が終わらなくなることが増えたりはする。証明の効率を落とさないためには、何らかのヒューリスティックスを用いて忘却する知識を選択する必要があるだろう。(現実装では特にそのような工夫は行っていない。)

[2] 脳においては、海馬が体験をエピソード記憶として瞬時に記憶し、それをもとに大脳皮質が時間をかけて意味記憶を形成すると考えられている [9]。

```
1: procedure PROVE(p)
2:     s ← {}      # 証明済み命題の集合
3:     while p ∉ s do
4:         前提がすべて証明済みである推論規則を１つ選択し、その結論を s に追加する。
```

図 2: 命題 p を証明する極めてシンプルなアルゴリズムの疑似コード。Pro5Lang ではこの証明済み命題集合 s を宣言的知識の集合と見なしている。ただし実際の Pro5Lang の証明探索アルゴリズムはより複雑で効率的である。

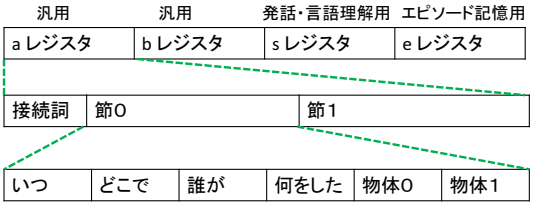

図 3: 現バージョンの Pro5Lang のレジスタ構成。すべてのレジスタは同じ構造を持つ固定長のベクトルである。１つのレジスタは複文程度の情報を持つ命題を表現するのに適した構造を持つ。

3.4 Pro5Lang の概要

我々が RGoal を用いて構築中のプログラム合成システムでは、強化学習における状態行動対をパターンを用いた表現に圧縮・抽象化したものを行動ルール集合と呼ぶ [2][5]。この行動ルール集合が Pro5Lang のプログラムである。行動ルールは強化学習によって学習した価値に応じた確率で選択され、メモリの参照・更新や環境に対する行動出力を行う。学習が進めば、エージェントはセンサーからの入力に基づいて、報酬を最大化するような行動を選択するようになる。

この言語は２つの意味で確率的な言語である。第１に、環境の変化が確率的であることを想定した設計になっている。第２に、実行する行動ルールはその価値に応じた確率によって選択されるため、この言語自身の振る舞いも確率的である。

Pro5Lang の機械語に似た特徴としては、(1) すべての命令（行動ルール）とデータ（証明済み命題）が統一された形式を持つ固定長の整数ベクトルで表現される、(2) 少数のレジスタと大容量のメモリを用いて情報処理を行う、(3) 固定した回路（論理回路または神経回路）で実現可能なほど言語仕様がシンプルである、といった点がある。

Pro5Lang の詳細は以下の節で述べていく。

3.5 レジスタ構成と命題の表現

図 3 は今回実装した Pro5Lang プロトタイプシステムのレジスタ構成である。a,b,s,e という名前の付いた４つのレジスタがある。a,b レジスタは汎用レジスタ、s は発話・言語理解用のレジスタ、e はエピソード記憶想起用のレジスタである。４つのレジスタは同一の構造であり、複文程度の情報を持つ命題を表現するのに適した構造を持つ。（レジスタの数やレジスタ内に含まれる節の数は今後拡張される可能性がある。）

今回の実装では時間・場所・物体などを表現するためにシンボルを用いるが、将来の実装ではすべて固定長の数値ベクトルに置き換える予定である。つまり、ある概念の内容を埋め込んだ特徴ベクトルをその概念のＩＤとして用いる。数値ベクトル表現は深層学習のような手法で環境から獲得することができるため、環境に接地していると言える。接続詞については環境に直接は接地しないが、[4] で述べた方法で接続詞を含む命題を扱う推論規則の価値を学習できるため、間接的に環境に接地できると考えている。

Pro5Lang には記憶の場所を指し示すポインタのようなものはない。概念のＩＤを用いた連想記憶により、ポインタがなくても必要な情報処理が十分に行えるだろうと現在のところは考えている。ポインタを使わないことは、記憶域管理（不要な記憶の忘却及び、エピソード記憶から意味記憶への記憶の定着）を容易にする上で必須であるように思われる。

証明の健全性を容易に保てるようにすために、命題は解釈する文脈によらずに真偽値が決められるような自己完結的な表現になっている必要がある。そのため、時刻と場所は絶対座標を用いた表現（もしくは絶対座標に変換可能な情報を持った表現）である必要があるだろう。事実と異なる仮想的な状況（反実仮想）も扱えるようにするため、特殊な時刻表現の導入も必要だろう。また、記号的に表現可能な文脈表現だけでなく、言語化し得ないような非記号的な文脈の情報も、ある程度の長さを持った数値ベクトルの形で命題の表現に含めるべきだろう。これらについては現在の実装では扱っていないが、証明の健全性という強力な設計指針にしたがって自然に設計可能であると考えている。

3.6 テストプログラム用 DSL

Pro5Lang がヒトが解くべきタスクに対する十分な記述力を持っているのか、プログラム合成の対象言語として適切な性質を持っているか、などを検証する目的で、

```
Pro5 プログラム  = 宣言的知識* 行動ルール*
宣言的知識       = k(e(命題));
行動ルール       = rule(状態, サブゴール, 行動);
状態             = w(レジスタ*)
レジスタ         = a(命題) | b(命題) | s(命題) | e(命題)
命題             = 接続詞, 節, 節 | 略記節
節               = c(時間, 場所, 物体, 述語, 物体, 物体) | c(略記節)
略記節           = 述語 | 物体 述語
サブゴール       = 状態
行動             = call(レジスタ) | set(レジスタ) | recall(e(命題))
接続詞, 時間, 場所, 述語, 物体       = シンボル | 変数 | 整数 | "PLS" | "__"
シンボルは大文字で始まる識別子。
変数は小文字で始まる識別子。
```

図 4: テストプログラム用 DSL の文法。“*”は 0 個以上の繰り返しを表す。

Pro5Lang のプロトタイプシステムを実装した。Java 言語のソースコード中に埋め込める DSL (Domain-Specific Language) として Pro5Lang を実装し、この DSL で様々なテストプログラムを手で書いて動かしてみることにより、処理系の動作検証と言語仕様の妥当性の検証を現在進めている。

テストプログラム用 DSL の BNF (Backus–Naur form) に似た記法による文法を図 4 に示す。テストプログラムは宣言的知識の定義と、行動ルール（手続き的知識）の定義から構成される。（どちらの知識も本来は AGI エージェントが自分自身の経験を通じて蓄積していくものだが、 DSL ではテストが目的なのですべて人間が手で与える。）

宣言的知識は k(e(命題)) という形で定義する。定義された命題は宣言的記憶の初期値に追加される。

行動ルールは rule(s,g,a) という形で定義する。エージェントは、現在のレジスタおよびサブゴールの値とマッチするパターン (s,g) を持つ行動ルールを非決定論的に１つ選択し、その行動 a を実行する。もしマッチする行動ルールが１つもなければ（Prolog 言語の fail に相当）、状態をリセットして最初から証明をやり直す。（詳細は [4] を参照。）

現実装では、エージェントが取り得る行動は脳内アクションのみである。（環境とのインタラクションは未実装である。）具体的には call, set, recall のうちどれかであり、それぞれサブルーチン呼び出し、レジスタの値の更新、宣言的知識の想起を意味する。現実装における各行動の具体的な動作は以下の通りである。

- call(g) はサブルーチン呼び出しであり、サブゴールを g に設定する。（詳細は [4] を参照。）
- set(r(p)) はレジスタ r (ただし r は a,b,s,e のいずれか) の値を p に更新する。指定したレジスタ以外のレジスタの値は変化しない。もし a レジスタが指定されたときは、値 p を証明済み命題集合に追加する。
- recall(e(q)) は、宣言的知識の中から検索クエリ q により想起されるものを非決定論的に１つ選択し、結果を e レジスタに代入する。想起すべき宣言的知識が１つもない場合（想起の fail と呼ぶ）は、状態をリセットして最初から証明をやり直す。想起の仕方の詳細は次節で述べる。

3.7 宣言的記憶の想起

宣言的記憶の想起に関しても、証明の健全性を保つという設計指針を重視する。証明の健全性は、想起する内容もまた証明済み命題に限定することで容易に達成される。宣言的記憶の中身は証明済み命題に限定されているので、記憶されている内容をそのまま読みだせばよい。逆に言えば、宣言的記憶の中身を改変して読みだすような言語仕様は排除される。

エピソードのパターン補完は、Pro5Lang では以下のように実装される。命題は固定長の整数ベクトルで表現されるが、その要素には unknown を意味する特別な値（現実装では 0 としている）が入り得る。パターン補完は、検索クエリの値に含まれる unknown な要素を、宣言的記憶側の任意の値と一致すると見なすことで実現される[3]。例えば「きのう兄がチョコレートを食べた」を意味するエピソード記憶は、「きのう <unknown> が <unknown> を食べた」を意味する検索クエリを使って想起することができる。

[3]宣言的記憶側の unknown な要素を検索クエリ側の値で補完して読みだすことはない。それは健全性を損なう。ヒトは過去の不確かな記憶をその後の経験や推論によって具体化することはあるが、それは強化学習によって健全性が保証される行動ルールが行うのであって、想起の機構が行うのではない、と解釈する。

現在の Pro5Lang では、検索クエリに “PLS” という特別な値を書くことができる。これは unknown 以外の値にのみマッチする。例えば「きのう PLS が PLS を食べた」という内容の検索クエリを書くことができる。

Pro5Lang の意味記憶は、多数の証明済み命題を圧縮・抽象化して１つの命題のパターンの形で表現したもので、その表現は任意の値とマッチする変数やワイルドカード “_” を含みうる。（4.3 節で例を示す。）意味記憶が想起される時は、変数の値を検索クエリが持つ値に置き換えたものが想起される。意味記憶が過汎化した間違った知識でない限りは、これにより健全性が失われることはない。

複数の宣言的記憶の候補があったときの想起確率をどう決めればよいのかなどの細かな仕様は証明の健全性の要請だけからは確定できず、設計に自由度がある。設計によって、プログラムの合成や実行の性能が変わる。例えば、情報量の多い知識や新しい知識ほど高い確率で想起するといったヒューリスティックスが考えられるが、何が最適かは今後検討していく。

4　テストプログラム例

4.1　推論規則 P,Q ⊢ R

図 5 は、推論規則 P,Q ⊢ R を実現する行動ルール集合の例である。行１で状態として w() が指定されているが、これは任意の状態にマッチするパターンであり、まず行１が最初に実行される。サブルーチン呼び出し call(a(P)) の実行によって a レジスタの値は P になるので次には行２が実行され、さらに行３，４が実行されて、サブゴール a(R) が達成される（a レジスタの値が R になる）のでサブルーチンの呼び出し元に return する。行３で命題 P をわざわざ想起する必要があるのは、行２で呼び出すサブルーチン内でレジスタの値が破壊され、P が証明済みかどうかわからなくなるからである[4]。

4.2　環境の隠れた状態の推定

図 6 は、家に何か（食べるものが）があるかどうかを推論するプログラムの例である。これは以前の文献 [3] で用いた例と同じである。

行１～３ではエピソード記憶を直接記述して与えている。（本来はエージェントが経験から得るべき知識である。）

行４は同じ記述の繰り返しを避けるためのマクロ定義のようなもので、それ以降に現れる g はすべて代入された式に置き換えられる。

行５～８は g で定義されるサブゴールを証明するためのサブルーチン定義で、前節の図 5 とほぼ同じ構造をしている。

行５の recall 命令は、行１と行２で与えているエピソードのどちらかをランダムに選択する。もし行２の「きのうスナックがあった」が選択されたら、そのあとの行７の recall 命令による「きょう兄がスナックを食べていない」というエピソードの想起が fail し、証明を最初からやり直す。もし行５の recall 命令で行１の「きのうチョコレートがあった」が選択された場合は行７で「きのう兄がチョコレートを食べていない」が想起され、最終的に「きょう家にチョコレートがある」が証明されて、このサブルーチンのサブゴール g が達成される。

4.3　意味記憶を用いた推論

図 7 は意味記憶を用いて推論を行うプログラム例である。

行１は「ソクラテスは人間である」、行２は「人間は死ぬ」という内容を表す意味記憶を定義している[5]。行７～１０は P,P → Q ⊢ Q という汎用の推論規則を定義している。

ゴールとして「ソクラテスは死ぬ」を表す次の命題をセットしてこのプログラムを実行する。

```
a(0, 0, Socrates, Is, Mortal, 0)
```

最短[6]では行７→８→１２→１３→９→１０の順で実行が進み、ゴールが証明される。

証明の過程で何が推論されたかを日本語でわかりやすく説明すると以下のようになる。

1. 「ソクラテスは死ぬ」を証明するために「X ならば「ソクラテスは死ぬ」」という知識を思い出してみる。（行７）
2. 「「ソクラテスは人間」ならば「ソクラテスは死ぬ」」が想起できたら、「ソクラテスは人間」を証明してみる。（行８）
3. 「ソクラテスは人間」が証明できたら、再度「「ソクラテスは人間」ならば「ソクラテスは死ぬ」」が成り立つかどうかを想起してみる。（行９）

[4] 以前の実装方法（[4] の実行例５）では何らかの原因で述語 Q が成り立ってしまった場合、P が成り立っていないにも関わらず R が成り立つと推論してしまう可能性があるので、状況によっては好ましくない。

[5] 意味記憶における「いつ」「どこに」のスロットに何を入れるべきなのかは再考の余地があるが、この実装では 0 を入れている。

[6] 最初に行７ではなく行１２が選択され recall が fail することもある。

```
1: rule(w(),          w(a(R)), call(a(P)));    // まず P を証明
2: rule(w(a(P)),      w(a(R)), call(a(Q)));    // P の次は Q を証明
3: rule(w(a(Q)),      w(a(R)), recall(e(P)));  // Q が証明済みなら P が証明済みかどうか想起
4: rule(w(a(Q),e(P)), w(a(R)), set(a(R)));     // P,Q が証明済みならば R は成り立つ
```

図 5: P,Q ⊢ R という推論規則にしたがって R を証明するサブルーチンの例。再帰的に P, Q を証明するサブルーチンを呼び出している。

```
1: k(e(Yesterday, Home, Chocolate, Exists, 0,  0));  // きのうチョコレートがあった
2: k(e(Yesterday, Home, Snack, Exists, 0,  0));      // きのうスナックがあった
3: k(e(Today, Home, Brother, NotEat, Chocolate, 0)); // 兄はチョコレートを食べてない

4: g = w(a(Today, Home, PLS, Exists, 0, 0));         // サブゴール：「きょう家に PLS がある」
5: rule(w(), g, recall(e(Yesterday, Home, PLS, Exists, 0,  0)));
6: rule(w(e(Yesterday, Home, x, Exists, 0,  0)),
        g, set(a(Yesterday, Home, x, Exists, 0,  0)));
7: rule(w(a(Yesterday, Home, x, Exists, 0,  0)),
        g, recall(e(Today, Home, Brother, NotEat, x, 0)));
8: rule(w(a(Yesterday, Home, x, Exists, 0,  0),      // きのう x があり、
          e(Today, Home, Brother, NotEat, x, 0)),    // 兄が x を食べていないならば、
        g, set(a(Today, Home, x, Exists, 0, 0)));    // x がある
```

図 6: エピソード記憶（行１～３）をもとに、家に何か（食べるものが）があるかどうかを推論するサブルーチン（行５～８）の例。図 5 と基本的な構造は同じである。“PLS” は 0 以外の値にマッチする特殊なパターン、小文字の “x” は任意の値とマッチする変数である。本来はエピソード記憶内の時間・場所は絶対座標で表現すべきである（そうでないと想起した命題が必ずしも真にならなくなる）が、ここではわかりやすさのために Today, Yesterday を使っている。

```
 1: k(e(0, 0, Socrates, Is, AMan, 0));      // ソクラテスは人間である
 2: k(e(If, c(0, 0, x, Is, AMan, 0), c(0, 0, x, Is, Mortal, 0))); // 人間は死ぬ

 3: eifxy = e(If, c(x1,x2,x3,x4,x5,x6), c(y1,y2,y3,y4,y5,y6));
 4: eif0y = e(If, c(__,__,__,__,__,__), c(y1,y2,y3,y4,y5,y6));
 5: ax = a(x1,x2,x3,x4,x5,x6);
 6: ay = a(y1,y2,y3,y4,y5,y6);
 7: rule(w(),       w(ay), recall(eif0y));  // y を証明するために x → y を想起
 8: rule(w(eifxy),  w(ay), call(ax));       // x → y なら y を証明するために x を証明
 9: rule(w(ax),     w(ay), recall(eifxy));  // x なら y を証明するために x → y を想起
10: rule(w(ax, eifxy), w(ay), set(ay));     // x, x → y なら y が成り立つ

11: ex = e(x1,x2,x3,x4,x5,x6);
12: rule(w(),   w(ax), recall(ex));
13: rule(w(ex), w(ax), set(ax));
```

図 7: 意味記憶（行１～２）と汎用の推論規則（行７～１０）を用いて「ソクラテスは死ぬ」を推論するプログラムの例。

4. 「ソクラテスは人間」と「「ソクラテスは人間」ならば「ソクラテスは死ぬ」」が成り立つので、「ソクラテスは死ぬ」が成り立つと結論付ける。（行１０）

4.4 命題の合成・分解

Pro5Lang が扱う情報の単位を複文程度としたため、前節で述べたような If を使った推論が表現可能になったが、命題の合成・分解の処理も表現可能になる。A,B ⊢ A ∧ B は A, B という２つの論理式を A ∧ B という１つの論理式に合成する推論規則だが、同様の内容を Pro5Lang の行動ルールでは次のように表現する。

```
rule(w(a(A),b(B)), g, set(a(And, c(A), c(B))));
```

逆に A ∧ B ⊢ A と A ∧ B ⊢ B という推論規則は次のように書ける。

```
rule(w(a(And, c(A), c(B))), g, set(a(A)));
rule(w(a(And, c(A), c(B))), g, set(a(B)));
```

このことは、関連したエピソードが持つ情報を１つの命題にまとめたり、逆に１つの命題を複数の細かい命題に分解したりするときに利用できる。

4.5 文の意味

現在のところ未実装であるが、自然言語の発話内容の意味もレジスタを使って表現する予定である。

他者の発話内容は脳の言語野に相当する機構によって意味解析され、結果は s レジスタに入れられるものとする。Pro5Lang の設計指針に従うならば、発話内容もまた証明済み命題で表現されなければならない。他者 A が S という内容の発話をしたならば、センサーと言語野が認識したその事実を「いまここで A が S と発話した。」という命題で表現すればよい。（発話内容 S は命題ではあるが、証明済み命題ではない点に注意されたい。）発話内容を認識したあとのエージェントの動作はエージェントの目的による。例えば「ハサミの場所の知りたい」という目的を持っているときに母が「ハサミは台所にある」と言ったならば、エージェントは「ハサミは台所にある」という命題が証明されたと信じるものとすると、そのようなエージェントの振る舞いを実現する汎用的な行動ルールの一例は以下のようになる[7]。

```
rule(w(s(ThatO1, c(Now, Here, Mother, Said, 0, 0),
                 c(Now, x, Scissors, Exists, 0))),
     w(a(Now, PLS, Scissors, Exists, 0, 0)),
     set(a(Now, x, Scissors, Exists, 0, 0)));
```

ここでは「母が「ハサミは x にある」と言った」という一種の埋め込み文は、ThatO1 という特殊な接続詞を用いて表現している。これは節 0 の１つめの目的語が指すものが節 1 の内容であることを示している。この行動ルールには Now, Mother, Exists といった定数が埋め込まれているが、これらも変数を使って抽象化すれば、行動ルールの汎用性は大きく高まる。比較的少数の汎用性の高い行動ルールの組み合わせにより、様々な言語活動が実現できるだろう。

この枠組みは「言語を理解するとはどういうことか」という問いに対する１つの答えを与える。このエージェントにとって言語の理解とは、その時のエージェント自身の目的に応じて、他者の発話内容に対し自分自身が信じる推論規則を適用し、様々な命題を証明し、その結果を記憶し、目的の達成に役立てることである。

つまり、エージェントは対話を目的達成の道具として合目的的に使用することになる。目的達成の手段は無数にあるかもしれないが、強化学習によって行動ルールの価値が学習される [4] ため、十分な学習の末には、簡潔で誤解の起きにくい発話・言語理解ができるようになることが期待できる。

命令文、疑問文、感嘆文などや、他者の意図なども証明済み命題に符号化して表現することで、あらゆる言語理解・発話計画を報酬最大化を目的とする証明探索に帰着させて扱えるようになる。

以上に述べたように発話と言語理解のための道具立てはすでに整っている。行動ルールを自律的に獲得するための道筋もついている [7]。現在の実装を少し拡張するだけで、言語を通じた行動ルールの獲得も実現できると考えており、今後設計・実装を進めていく。

5 正しい宣言的知識の選別方法の問題

正しい手続き的知識（行動ルール）の選別は [4] で述べた機構により行えるが、正しい宣言的知識を選別する方法は自明ではない。宣言的知識の想起は recall 命令を持つ行動ルールで行われるので、基本的には強化学習により、間違った知識を想起する行動ルールは実行されなくなる。しかし、検索クエリは抽象的なので、うまくいかないことがある。例えば「フランスの首都は PLS である」という検索クエリは「フランスの首都はパリである」と「フランスの首都はロンドンである」の両方に等しくマッチするので、間違った知識の選別には役立たない。これはパターンを用いて行動価値関数を圧縮することで価値の精度低下 [2] が生じることが原因であると解釈できる。この問題の解決策の１つとして、Dyna-2 アーキテクチャ[10] が持つような一時メモリを導入することで行動価値関数の精度を上げる方法が考えられる。想起した結果の値を使って、パターンではない具体的な値表現だけから構成される行動ルール rule(s,g,recall(p)) を一時メモリに入れて、その価値を学習する。そうすれば、間違った知識を想起する一時メモリ上の行動ルールの価値が下がるので

[7] より頑健な動作にするためには、「自信のありそうな言い方だったかどうか」といった、ニューラルネットが認識する非記号的な特徴も判断材料に入れて推論すべきであろう。

間違った知識の使用頻度も下がる。このアイデアの実装と動作確認は将来の課題である。

6 関連研究

BRA 駆動開発 [11] は脳の各部品の機能を再現するアーキテクチャを神経科学的知見と整合性を持たせて構築する開発方法論であり、これに基づいた海馬モデルの構築も試みられている [12]。一方我々の研究構想 [7] では脳型 AGI の本質部分のデモを極力早期に動かそうとしており、BRA 駆動開発とは相補的な関係にあると思われる。

強化学習アーキテクチャにはエピソード記憶と似た機構を持つものがある。DQN [13] の replay buffer は学習サンプルの相関を減らす experience replay を行うことを目的として、エージェントの経験を記憶する。Dyna-2 [10] の一時メモリは短い時間スケールの間の行動価値関数を記憶するもので、一般的な状況における行動価値関数を学習する永続メモリと組み合わせて使用する。これらは本稿で提案するエピソード記憶とは、役割も記憶する情報も本質的に異なる。海馬の役割は１つとは限らず、これらすべての役割を果たしている可能性があると考えている。

プログラム合成の先行研究は多いが、例えば以下のものがある。

DNC [14] はニューラルネットワークでできた計算機で、連想記憶に似た機能を持った外部メモリを持つ。教師あり学習または強化学習によってプログラム合成を行う。プログラムはネットワークの重みが表現する。

DreamCoder [15] は、 Lisp, LOGO, 物理法則を表す数式など、木構造を持つ広範囲の言語を合成対象とする。動的計画法を用いて、候補となるプログラムを表現する木を有望なものから順に生成する点などに工夫がある。

LEAPS [16] はプログラムを連続的な空間に埋め込んで探索することで効率的な合成を行うシステムである。合成対象言語は単純であり、メモリの読み書きの機構を持たない。

AlphaCode [17] は機械翻訳技術を応用して、英語で書かれた競技プログラムの問題文から仕様を満たすプログラムを合成するシステムである。C++, Python などのフルセットの言語を合成対象とするが、現時点では合成の対象が競技プログラムに限定されているため、より複雑なプログラムの合成に応用できるかは明らかではない。

7 まとめ

プログラミング言語 Pro5Lang の宣言的記憶機構について述べた。Pro5Lang は、脳型 AGI アーキテクチャの中核技術の１つとして我々が設計・実装を進めているプログラム合成システムの合成対象言語である。この機構は証明の健全性という数理論理学的な要請を重要な設計指針としつつ、ヒトのエピソード記憶に関する知見をヒントにして設計された。Pro5Lang の情報処理の最小単位は「いつ・どこで」という文脈の情報を含み自己完結的に意味を持つため、データ構造が破綻しにくい、記憶域管理が容易になるといった、合成対象言語として有望な性質を持っている。

Pro5Lang のプロトタイプを実装し、いくつかのテストプログラムを手で書いて動かしてみることで、ヒトが日常的に行うタスクを解く合成対象言語として十分な記述力を持ちそうであることを確認した。また、今後拡張すべき点を明らかにした。

提案する機構は脳の宣言的記憶の機構の計算論的モデルの有望な候補でもある。実験で検証可能な様々な予言がこのモデルで行えると思われるがその検討は将来の課題である。

将来的にはエージェントが自分自身の経験や対話・模倣などを通じて知識（手続き的知識および宣言的知識）を自律的に獲得するシステムにすることを目指しているが、そのためにやるべきことはまだ多い。重要度の高い課題の１つにエージェントどうしの対話を通じた知識獲得の機構の実現があり、現在検討を進めている。

謝辞

本研究は JSPS 科研費 JP18K11488, JP18K18117 の助成を受けたものです。

参考文献

[1] Yuuji Ichisugi, Naoto Takahashi, Hidemoto Nakada, and Takashi Sano. Hierarchical reinforcement learning with unlimited recursive subroutine calls. In *Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning*, pp. 103–114, Cham, 2019.

[2] 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.

[3] 一杉裕志, 中田秀基, 高橋直人, 佐野崇. 階層型強化学習 RGoal を用いた記号推論の実現手法の検

討. 第 12 回 人工知能学会 汎用人工知能研究会 (SIG-AGI), 2019.

[4] 一杉裕志, 中田秀基, 高橋直人, 佐野崇. 推論規則の価値を階層型強化学習 RGoal を用いて学習する手法の提案. 第 14 回 人工知能学会 汎用人工知能研究会 (SIG-AGI), 2020.

[5] 一杉裕志, 中田秀基, 高橋直人, 佐野崇. 脳の自律的プログラム合成機構のモデルに向けて：２層ベイジアンネットによる記号処理命令の獲得・実行機構. 第 15 回 人工知能学会 汎用人工知能研究会 (SIG-AGI), 2020.

[6] 一杉裕志, 中田秀基, 高橋直人, 佐野崇. 物体操作に適したワーキングメモリを持つ汎用人工知能アーキテクチャの検討. 第 16 回 人工知能学会 汎用人工知能研究会 (SIG-AGI), 2020.

[7] 一杉裕志. 報酬最大化原理にもとづく脳型ＡＧＩアーキテクチャの構想. 第 18 回 人工知能学会 汎用人工知能研究会 (SIG-AGI), 2021.

[8] Edmund Rolls. The mechanisms for pattern completion and pattern separation in the hippocampus. *Frontiers in systems neuroscience*, Vol. 7, p. 74, 2013.

[9] Randall CO ' Reilly, Rajan Bhattacharyya, Michael D Howard, Nicholas Ketz. Complementary learning systems. *Cognitive science*, Vol. 38, No. 6, pp. 1229–1248, 2014.

[10] David Silver, Richard S Sutton, and Martin Müller. Sample-based learning and search with permanent and transient memories. In *Proceedings of the 25th international conference on Machine learning*, pp. 968–975, 2008.

[11] Hiroshi Yamakawa. The whole brain architecture approach: Accelerating the development of artificial general intelligence by referring to the brain. *Neural Networks*, Vol. 144, pp. 478–495, 2021.

[12] Akira Taniguchi, Ayako Fukawa, and Hiroshi Yamakawa. Hippocampal formation-inspired probabilistic generative model. *arXiv preprint arXiv:2103.07356*, 2021.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. Human-level control through deep reinforcement learning. *Nature*, Vol. 518, No. 7540, pp. 529–533, 2015.

[14] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, Vol. 538, No. 7626, pp. 471–476, 2016.

[15] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lucas Morales, Luke Hewitt, Armando Solar-Lezama, and Joshua B Tenenbaum. DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning. *arXiv preprint arXiv:2006.08381*, 2020.

[16] Dweep Kumarbhai Trivedi, Jesse Zhang, Shao-Hua Sun, and Joseph J Lim. Learning to synthesize programs as interpretable and generalizable policies. *Advances in Neural Information Processing Systems*, Vol. 34, , 2021.

[17] Yujia Li, et al. Competition-Level Code Generation with AlphaCode. preprint `https://storage.googleapis.com/deepmind-media/AlphaCode/competition_level_code_generation_with_alphacode.pdf`, 2022.