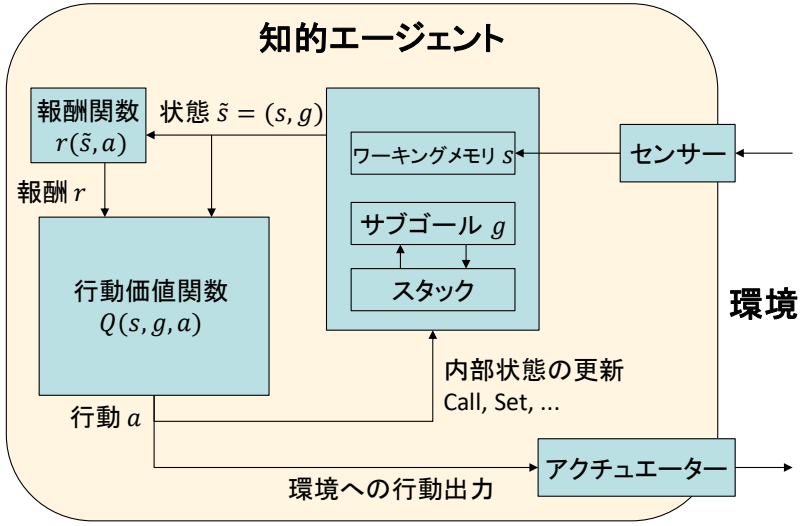
汎用人工知能のためのプログラム合成対象言語 Pro5Lang のエピソード記憶機構

第20回 汎用人工知能研究会(SIG-AGI)
2022-3-15
ー杉裕志、中田秀基、高橋直人、竹内泉（産総研）、
佐野崇（東洋大学）

中期的な研究構想[一杉 第18回汎用人工知能研究会 2021]

脳をヒントにして、再帰的強化学習、**プログラム合成**、生成モデルの3つを
中核技術とした脳型AGIアーキテクチャの構築を目指す

圧縮表現された
行動価値関数
＝
報酬を最大化する
プログラム



再帰的強化学習 RGoal のアーキテクチャ

十分な性能を持つプログラム合成システムは AGI と見なせるだろう

性能を上げるためには**合成対象言語**を適切に設計する必要がある

発表の概要

- 背景：解決すべき問題
  - 合成対象言語のメモリ機構
  - エピソード記憶の計算論的役割
- 合成対象言語 Pro5Lang ＝論理型言語＋機械語
- 提案するメモリ機構
  - 「エピソードは証明済み命題である」という仮説に基づいて設計
- テストプログラムによる記述力の予備的な検証
- まとめ

プログラム合成

- プログラム合成は AGI 実現への有望なアプローチ
  - AIXI [Hutter 2000], MagicHaskeller [Katayama 2008, 2018], DNC [Graves+ 2016], DreamCoder [Ellis+ 2020], LEAPS [Trivedi+ 2021], AlphaCode [Yujia+ 2022]
- 合成対象のプログラミング言語の性質が、性能に大きく影響
  - 単純な言語：合成しやすいが低機能
  - 複雑な言語：高機能だが合成しにくい

合成しやすく、かつ十分な表現力を持つ合成対象言語の設計が必要

- プログラムが高い表現力を持つためには大容量メモリが必須
  → エピソード記憶

解くべきタスクに特化させて性能を上げる

- ヒトの脳が日常的に解くべきタスク:
  - 物体操作:
    - 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:
    - 「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
    - 事実の伝達、要求、命令、約束、謝罪、感謝など

これらのタスクに特化したメモリ構造を持たせることで、
プログラム合成の性能の向上を図る必要がある

記憶の種類

- **手続き的記憶**
  - 運動技能など
    - 自転車に乗る方法、パズルの解き方
- **宣言的記憶**
  - **エピソード記憶**：「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
    - 「きょう，学校でテストがあった」
  - **意味記憶**：特定の時刻・場所に関係しない一般的な知識
    - 「地球は1年で太陽のまわりを一周する」

参考：「陳述記憶・非陳述記憶 - 脳科学辞典」
https://bsd.neuroinf.jp/wiki/陳述記憶・非陳述記憶
「意味記憶とは - コトバンク」,出典　ブリタニカ国際大百科事典 小項目事典ブリタニカ国際大百科事典
https://kotobank.jp/word/%E6%84%8F%E5%91%B3%E8%A8%98%E6%86%B6-164928

脳型AGIアーキテクチャの宣言的記憶をどう設計すればよい？

- 脳を模倣したAGIは宣言的記憶の機構を持つべき
- 仕様に関する数多くの疑問：
  - エピソードの想起することは過去の大脳皮質の発火パターンの再現？全体？一部？一部だとしたらどの部分？
  - 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？
  - ・・・
- **そもそも、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しない**

→ もしあれば脳の理解・脳型AGIの実現に大きく貢献

解決すべき問題

１．十分な表現力と合成のしやすさを合わせ持った合成対象言語のメモリ機構の設計が必要

２．宣言的記憶（エピソード記憶＋意味記憶）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が必要

これらの問題を解決するために合成対象言語 Pro5Lang のメモリ機構を設計した

合成対象言語 Pro5Lang

- 以前から設計・実装を進めている合成対象言語
  - 階層型強化学習における行動価値関数 Q(s,g,a) を圧縮表現したものがプログラム
- 「確率的証明済み命題処理プログラミング言語」
  “Probabilistic Proven-Proposition Processing Programming Language” = **Pro5Lang**
- Pro5Lang = 論理型言語＋機械語
  - **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
    プログラム ＝ 推論規則の集合
    プログラムの実行 ＝ 証明探索
    - **システムの振る舞いの理論的な見通しがよくなる**
  - **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
    - **神経回路での実現が容易**

推論規則と証明

推論規則: 前提 ⊢ 結論

例:

P と Q が成り立つときに R が成り立つと推論する推論規則:

$$P, Q \vdash R$$

前提なしに P が成り立つと推論する推論規則:

$$\vdash P$$

- 証明済みの命題に対し推論規則を適用することで、新たな命題が証明される
- 推論規則の集合がエージェントの世界モデル
  - 環境の隠れた状態の推定、未来の予測

極めて単純な証明探索アルゴリズムの例

```
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。
```

推論規則の集合:

$\vdash P$

$\vdash Q$

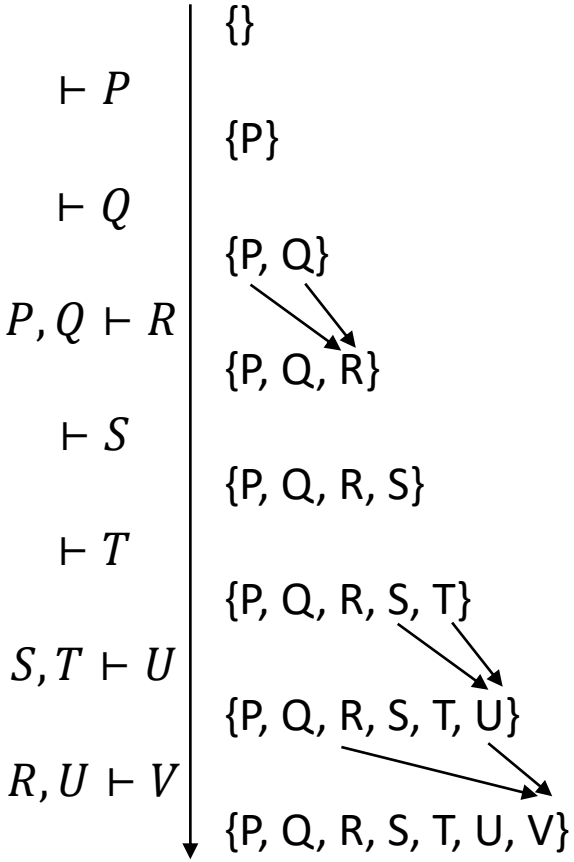
$\vdash S$

$\vdash T$

$P, Q \vdash R$

$S, T \vdash U$

$R, U \vdash V$

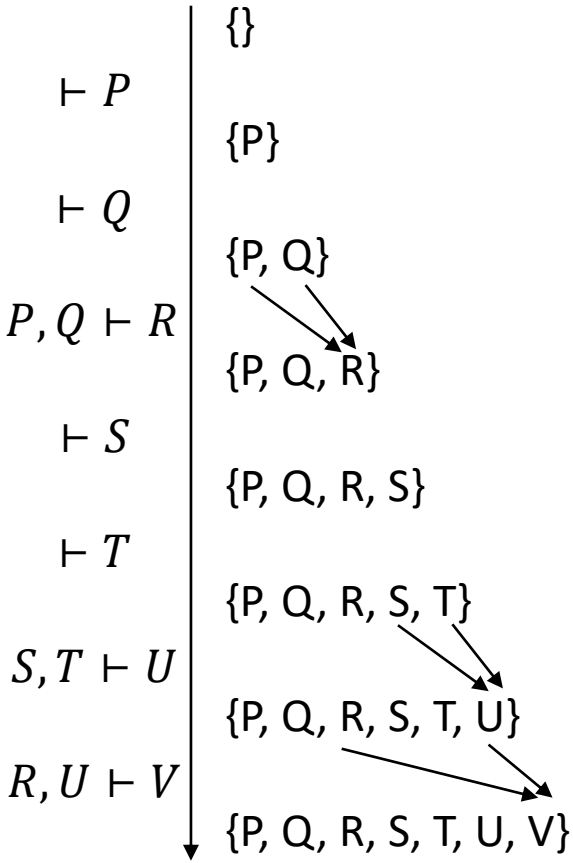


証明が進むにつれ、
証明済みの命題が
増えていく

(Pro5Lang の
証明アルゴリズムは
もっと複雑だが効率的)

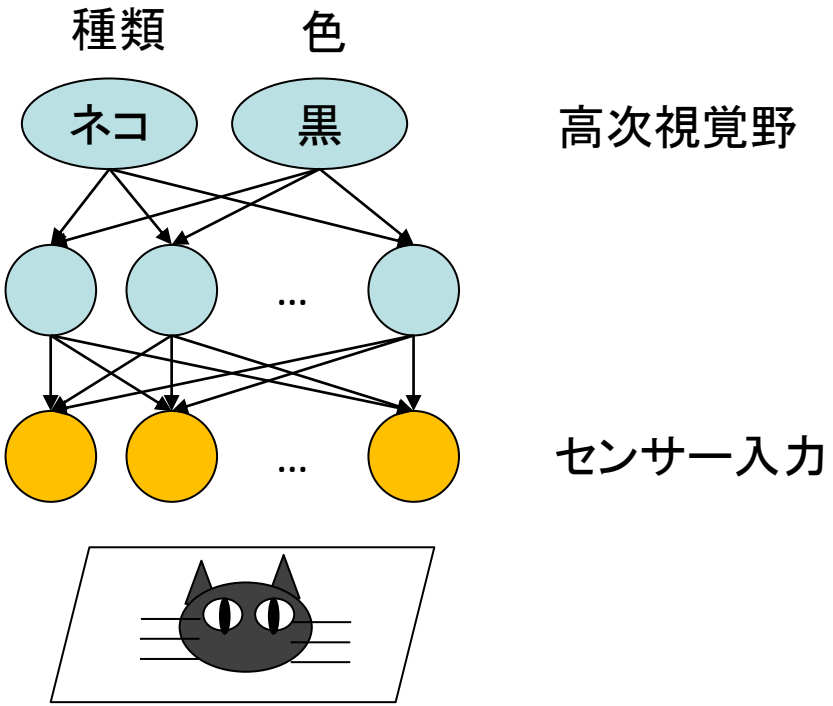
仮説:「エピソードは証明済み命題である」

- **この仮説を前提として Pro5Lang のメモリ機構を設計**
  - アーキテクチャ全体の見通しが良くなる
  - 細部の仕様の設計が容易になる
- エピソード記憶の役割:
  推論(証明)の前提として用いる目的で保持される
- 複数のエピソードを圧縮・抽象化したものが意味記憶



個人の体験も証明済み命題

　脳が黒いネコを認識した
＝「いまここに黒いネコがいる」という命題が成り立っていることが
センサー入力を証拠として証明された



テストプログラム例

テストプログラム用 DSL (Domain-Specific Language)

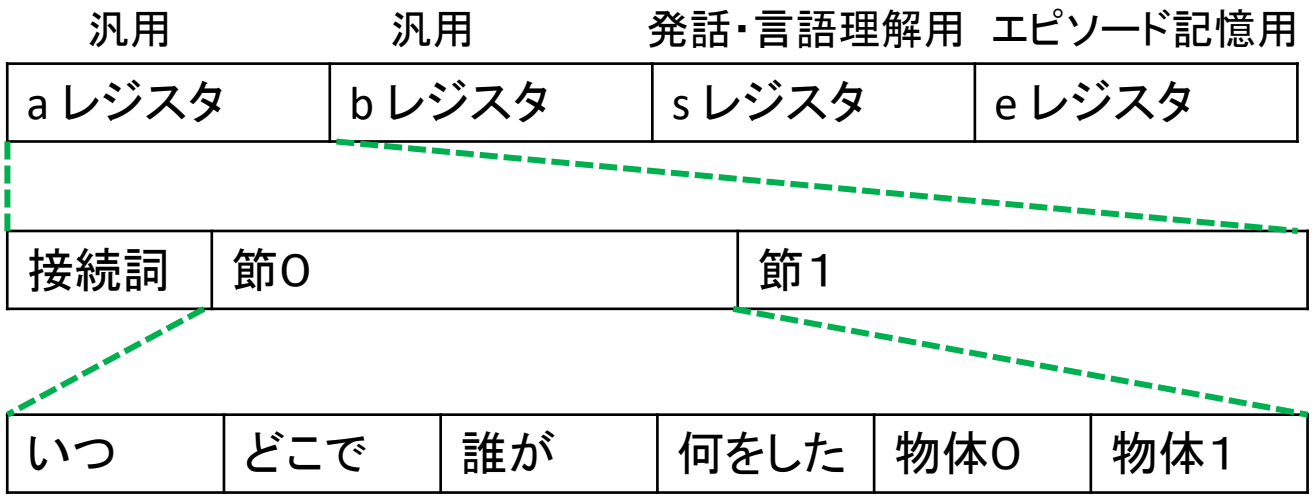
- Pro5Lang の設計の妥当性の検証が目的
  - 記述力、合成対象言語としての妥当性を**検証中**
- Pro5Lang の宣言的知識と行動ルール（手続き的知識）を**手で書いて**動作検証
  - 将来はエージェントが自分自身の経験から知識獲得
    [一杉+ 第14回汎用人工知能研究会 2020]
    [一杉+ 第15回汎用人工知能研究会 2020]

```
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" | "__ "
```

テストプログラム用 DSL の文法

Pro5Lang のメモリ機構

- 4つの**レジスタ**と大容量の**連想記憶装置（宣言的記憶）**に**証明済み命題のみ**を記憶
  - メモリの不整合が起きにくい → （おそらく）合成しやすい
- **1つの命題は複文程度の情報を表現**
  - 物体操作・推論・対話に特化した構造



現バージョンの Pro5Lang のレジスタ構成

宣言的記憶への記名と想起

記銘：a レジスタの値が更新されるたびに宣言的記憶に**自動的に保存**

想起：recall 命令

プログラム例：

```
// 推論規則 P,Q ├ R を使って R を証明するサブルーチン
1: rule(w(),           w(a(R)), call(a(P)));     // まず P を証明(P は自動的に保存される)
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
```

現在のレジスタの状態にマッチする行動ルールを1つ選んで実行

レジスタ a の値が P の時に実行される行動ルール
レジスタ a の値が Q の時に実行される行動ルール
レジスタ a, e の値が Q, P の時に実行される行動ルール

Q を証明するサブルーチンでレジスタの値が破壊される(行2)ので、
連想記憶に自動保存された P の値を想起(行3)してレジスタに入れなおしている

エピソード記憶を用いた、環境の隠れた状態の推論

エピソード記憶:

昨日はスナックがあった。
昨日はチョコレートがあった。
兄はチョコレートを食べていない。

推論規則:

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。

推論の目的:

今日何か(食べるものが)あるか?

```
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
```

(PLS は “unknown” でない具体的な値とマッチする特殊な値)

意味記憶を使う推論の例

```
 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));
```

「ソクラテスは人間である」と
「x が人間ならば x は死ぬ」という意味記憶と、
x, x → y ├ y という**汎用の推論規則**を用いて、
「ソクラテスは死ぬ」を推論

メモリ機構の設計指針：証明の健全性

- 健全性（正しい命題だけが証明されるという性質）
- **理想的な状況**（エージェントが正しい知識だけを持つ状況）で**健全性が成り立つように言語仕様を設計**
  - ただし、正しくない知識があってもそれなりに動く
- 言語仕様の詳細部分の設計に役立つ
  - エピソード想起の際のパターン補完, “unknown” の扱い
  - 命題が成り立つ文脈（「いつ・どこで」）の表現方法
  - 意味記憶の学習と想起
  - 宣言的知識の忘却

このシステムとGOFAI（昔の記号AI）との違い

- あくまで報酬最大化が目的
  - 強化学習により証明の効率が上がる（証明コスト最小化）
    - 考えるより聞いた方が速い、といった判断が可能
  - 状況に応じて精度より応答速度を優先して動作する
- 不完全な知識でもそれなりに動作
  - （完全性・健全性は理想的な状況でしか成り立たない。）
- 知識はエージェント自身が経験・模倣・言語を通じて獲得
- エージェントの動作も環境の変化も確率的

- GOFAI が行き詰まった原因のすべてが解決される見込み

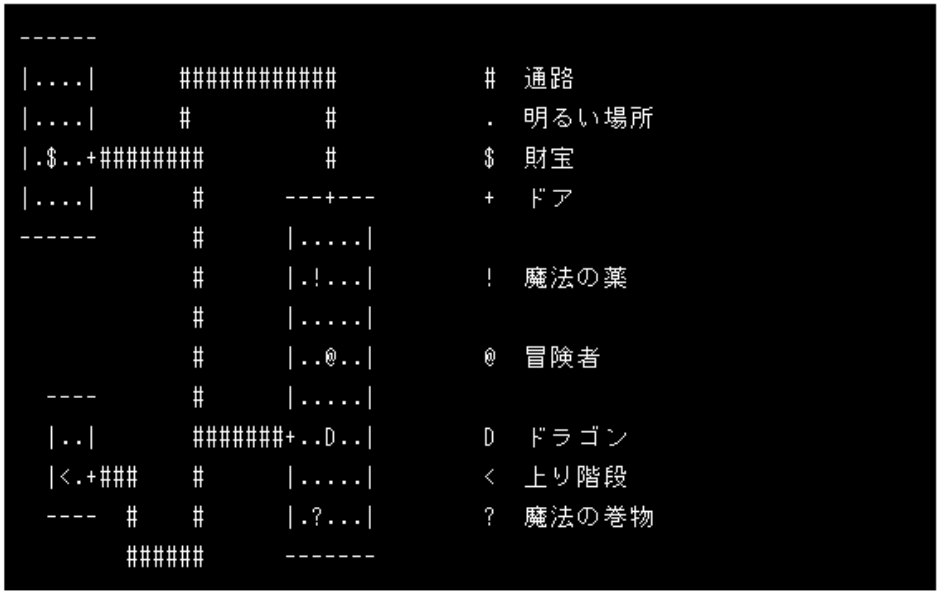
まとめと今後

- AGI のためのプログラム合成対象言語のメモリ機構を設計
  - エピソード記憶の計算論的モデルの候補
- 合成対象言語 Pro5Lang ＝論理型言語＋機械語
- 提案するメモリ機構
  - 仮説:「エピソードは証明済み命題である」
  - 証明の健全性を重視して設計
  - 物体操作・推論・対話のタスクに特化したレジスタ構成
- テストプログラムを手で書くことで表現力などを検証
  - プログラミング言語としての表現力は大きく向上
  - 合成対象言語として**現在のところ有望、今後も改良を続ける**
- 将来は、経験・対話・模倣などを通じて自律的に知識獲得

コメント・ご質問等よろしくお願いいたします

予備スライド

中期的目標：単純化された環境の中で対話や知識獲得する知的エージェントのデモ



ローグライクゲームの特徴：

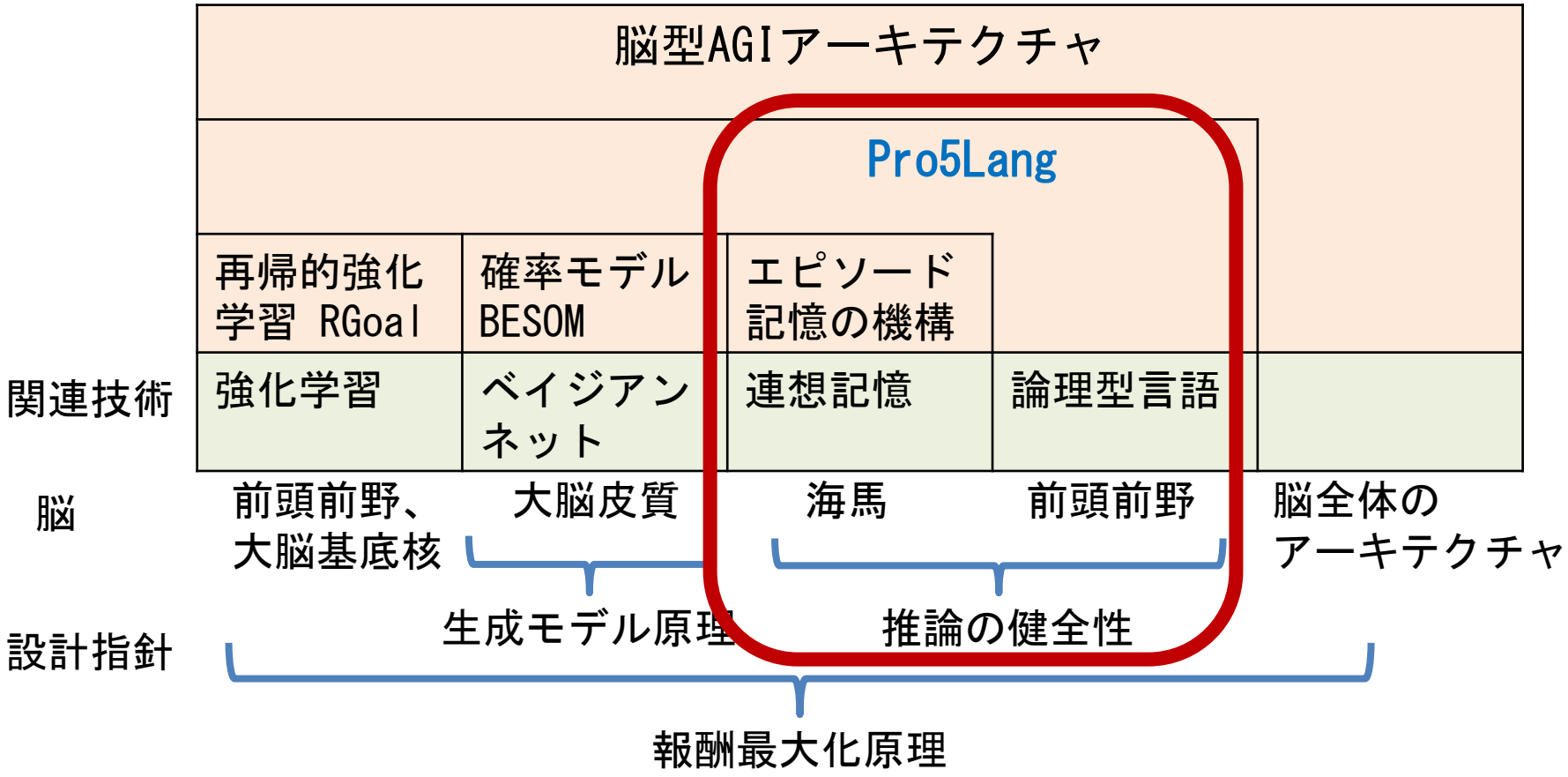
- 時間と空間が離散化
- マップ上のアイテムは記号化
- マップの大きさ、
  アイテムの数、
  物体同士の相互作用の数に制限がない
- 他者（敵）の動きが複雑
- プレイヤーは環境について経験から学習しなければならない

「ローグライクゲーム - Wikipedia」
https://ja.wikipedia.org/wiki/ローグライクゲーム

関連研究・関連分野

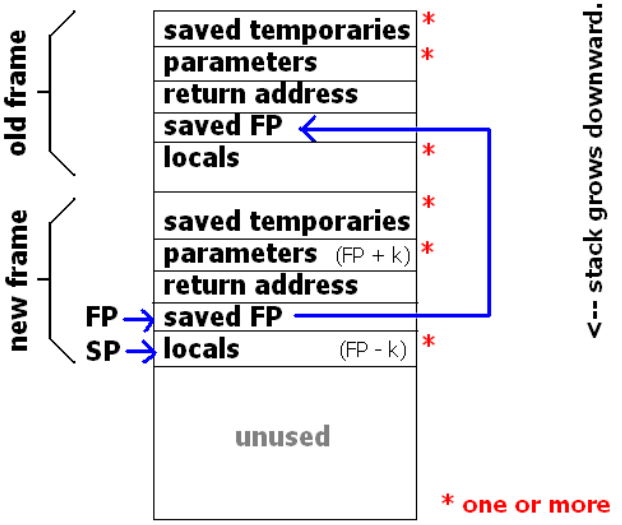
- 神経科学におけるエピソード記憶の計算論的モデル
  - パターン補完・パターン分離の神経回路モデル [Rolls 2013]
  - 海馬と大脳皮質の記憶の役割分担のモデル
    Complementary learning systems [O'Reilly+ 2014]
  - エピソード記憶の脳全体の情報処理における役割を説明する計算論的モデルは現時点では存在せず
- プログラム合成
  - DNC (Differentiable Neural Computer)[Graves+ 2016]
    - 連想記憶の機能も持つ外部メモリを持つ微分可能な計算機
- 数理論理学
  - 推論や証明の過程を定式化して研究の対象とする数学の一分野
  - 論理型言語 (Prolog など)の基礎

Pro5 言語の位置づけ



Pro5 言語にはスタックフレームがない

- **呼び出し側のレジスタの値の保存**
  - a レジスタの値は自動的にエピソード記憶に保存
- **サブルーチンのパラメタ**
  - サブルーチンが参照すべきパラメタは必ずワーキングメモリ上や環境にあるので、サブルーチンが自分の責任で観測
- **サブルーチン内のローカル変数**
  - エピソード記憶に保存
  - スコープがないのでおそらく再帰は苦手
- **リターンアドレス**に相当するサブゴール g は現在の実装ではスタックに保存
  - ただし RGoal にはスタックなし版もある
- スタックがなければ大域脱出（fail 、例外、割り込み）の処理が容易



プログラミング言語におけるスタックフレーム

「Stack frame - encyclopedia article - Citizendium」
https://en.citizendium.org/wiki/Stack_frame

データとプログラムの内部表現

プログラム例：

```
// 推論規則 P,Q ├ R を使って R を証明
1: rule(w(),          w(a(R)), call(a(P)));    // まず P を証明(P は自動的に保存される)
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
```

変換

内部的にはすべての宣言的知識・行動ルールは、固定長のベクトルで表現
（将来は BESOM Ver.4 を使って教師なし学習）

```
1:rule((__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,),
(Simple,Anytime,Anywhere,0,R,0,0,0,0,0,0,0,0,0,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,),
Call(Simple,Anytime,Anywhere,0,P,0,0,0,0,0,0,0,0,0,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,)).
2:rule((Simple,Anytime,Anywhere,0,P,0,0,0,0,0,0,0,0,0,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,),
(Simple,Anytime,Anywhere,0,R,0,0,0,0,0,0,0,0,0,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,),
Call(Simple,Anytime,Anywhere,0,Q,0,0,0,0,0,0,0,0,0,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,)).
3:rule((Simple,Anytime,Anywhere,0,Q,0,0,0,0,0,0,0,0,0,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,),
(Simple,Anytime,Anywhere,0,R,0,0,0,0,0,0,0,0,0,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,),
Recall(__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,Simple,Anytime,Anywhere,0,P,0,0,0,0,0,0,0,0,0,)).
4:rule((Simple,Anytime,Anywhere,0,Q,0,0,0,0,0,0,0,0,0,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,Simple,Anytime,Anywhere,0,P,0,0,0,0,0,0,0,0,0,),
(Simple,Anytime,Anywhere,0,R,0,0,0,0,0,0,0,0,0,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,__,),
Set(Simple,Anytime,Anywhere,0,R,0,0,0,0,0,0,0,0,0,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,=,)).
```

命題の合成･分解

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

// A∧B ⊢ A
rule(w(a(And, c(A), c(B))), g, set(a(A)));

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

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などに利用

文の意味（未実装）

他者の発話内容は脳の言語野に相当する機構によって
意味解析されて、結果が s レジスタに入るように実装する
予定

例：

目的：「ハサミの場所の知りたい」
「母が「ハサミは x にある」と言った」 ├「ハサミは x にある」

```
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)));
```

「言語を理解するとはどういういうことか」という問いに対する1つの答え：
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。

記号AIにおける様々な知識表現

- 手続き的知識
  - プロダクションルール（if-then ルール, ACT-R, Soar など）
- 宣言的知識
  - 知識グラフ（RDF など）
    - RDF: 主語（ノード）、述語（リンク）、目的語（ノード）の3つ組
  - フレーム表現

- 従来は手続き的知識は書くのは大変だが実行は速い（宣言的知識はその逆）などと言われた。