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)

2018-11-22

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

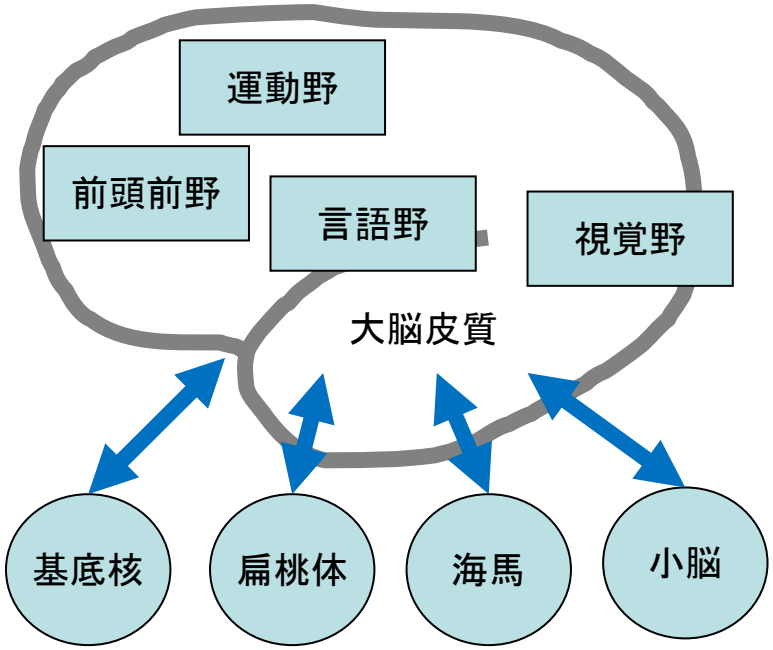
[1] National Institute of Advanced Industrial Science and Technology (AIST), AIRC

[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

私の研究の長期的目標

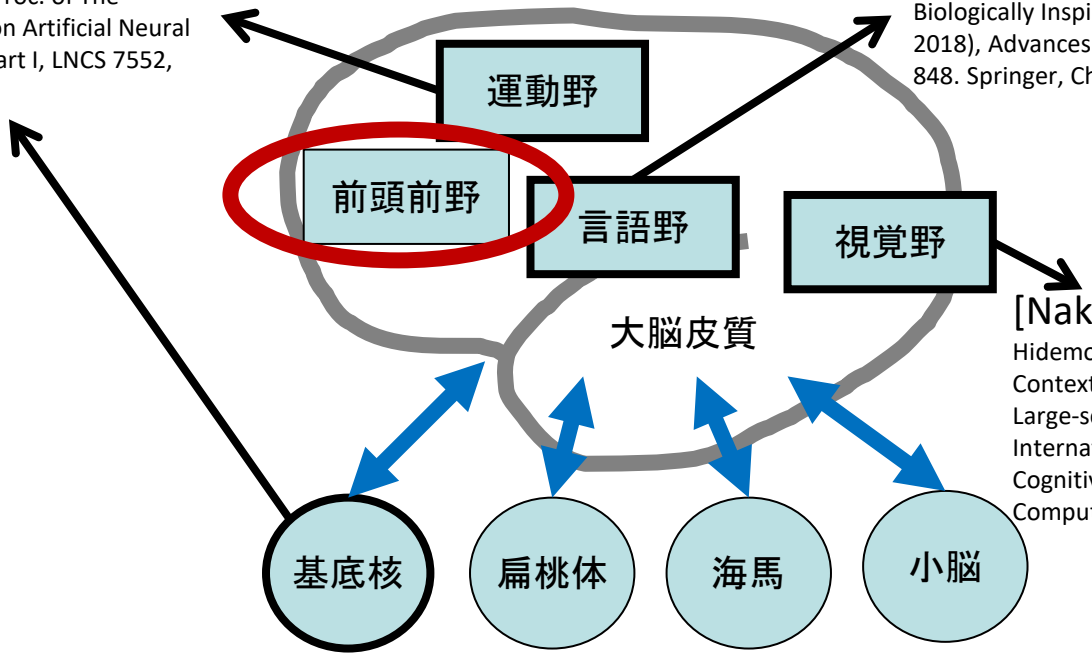
- 脳を模倣して「人間のような知能を持つ機械」を作る。（**脳のリバースエンジニアリング**）



現在の短期的目標：前頭前野周辺の計算モデルの構築を目指す

[Ichisugi, ICANN 2012]
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.

[Ichisugi and Takahashi, BICA 2018]
Ichisugi Y., Takahashi N., A Formal Model of the Mechanism of Semantic Analysis in the Brain. In: Biologically Inspired Cognitive Architectures 2018 (BICA 2018), Advances in Intelligent Systems and Computing, vol 848. Springer, Cham, pp.128--137, 2018.



[Nakada and Ichisugi, BICA 2017]
Hidemoto Nakada and Yuuji Ichisugi, Context-Dependent Robust Text Recognition using Large-scale Restricted Bayesian Network, In Proc. of International Conference on Biologically Inspired Cognitive Architectures (BICA 2017), Procedia Computer Science, Vol. 123, pp.314--320, 2018.

概要

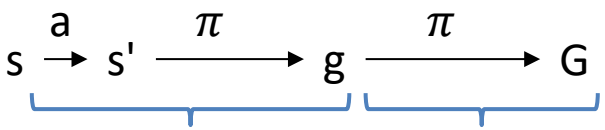
- 階層型強化学習 RGoal の行動価値関数のテーブル圧縮手法を提案
  - 大脳皮質による実現方法の1つの案を提案
  - 理想的に圧縮されたテーブルを手で書く**記述実験**により、提案手法を予備的評価
  - RGoal をプログラム合成システムと見なし、合成される「プログラム」の性質に関して考察

RGoal アーキテクチャ [一杉 2018年8月 汎用人工知能研究会]

- 多層の階層型強化学習アーキテクチャ
- エージェントは、各ステップで**行動する**か**サブルーチンを呼び出す**かのどちらかを選択
- ここではサブルーチン g は
  「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」
  - 例：はしごをかける
    （「はしごをかけた状態」をサブゴールとして行動する）

価値関数分解

- 関数 Q をサブゴール g の到着の前後で分解



$$Q_G^{\pi}((s,g),a) = \boldsymbol{M^{\pi}(s,g,a)} + V_G^{\pi}(g)$$

- 関数 M(s,g,a) は状態 s から g に到達するまでの報酬の総和の期待値
  - **おおもとのゴール G に依存しないため、異なるタスク間で共有できる。**
  - 共有により、マルチタスク環境で学習が速くなる。

RGoal のテーブル

- テーブル M(s,g,a) のインデックスの次元は環境の状態 s の次元の3倍
  - 素朴に実装するとテーブルのサイズが爆発
- このテーブルを**単一化の機構**を使って圧縮する手法を提案
  - 圧縮により学習パラメタが減り、学習速度・汎化性能がより向上するはず。

単一化(unification)とは

- 変数を含む2つの項が等しくなるように、変数の値を割り当てる操作。
- 計算機科学、理論言語学でよく使われる。
  - prolog 言語, 型推論など

例:

ベクトル1	ベクトル2	単一化の結果	変数の割り当て
(2,X)	(Y,3)	(2,3)	X=3,Y=2
(2,X)	(3,Y)	単一化失敗	-

テーブル圧縮の例

- パターンと値の組からなるルールの集合としてテーブルを表現

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

インデックスが 1 のとき 3.0
その他の時は 1.0

サイズが5のテーブルが2個のルールに圧縮

インデックスが2次元の場合の例

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

サイズ 5 x 5 = 25 のテーブルが4個のルールに圧縮

・ルールの順序には意味はないものとする。
・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。
・「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。

精度を落とした圧縮の例

- 一般に精度を落としてよければテーブルをより小さく圧縮できる。

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)	−∞

どのくらい圧縮すればよいかは、オンライン性能を最大化するためのハイパパラメタの1つ

単一化によるテーブル圧縮の脳内での実現の可能性

なぜ単一化？

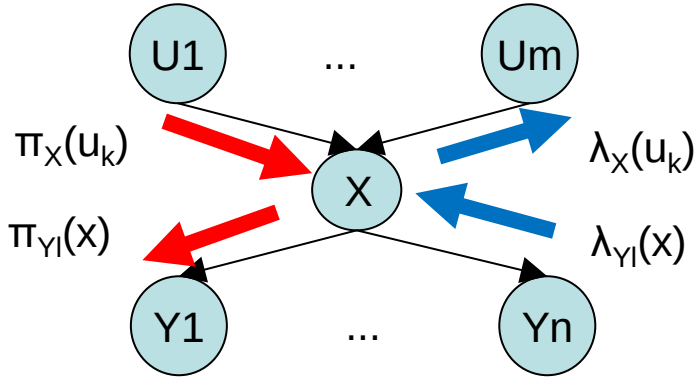
- **理論言語学における文法記述の枠組みの多くが単一化文法**
  - 単一化文法：単一化を使って構文解析を進めていく文法枠組みの総称
  - LFG, HPSG, CCG, TAG, ...

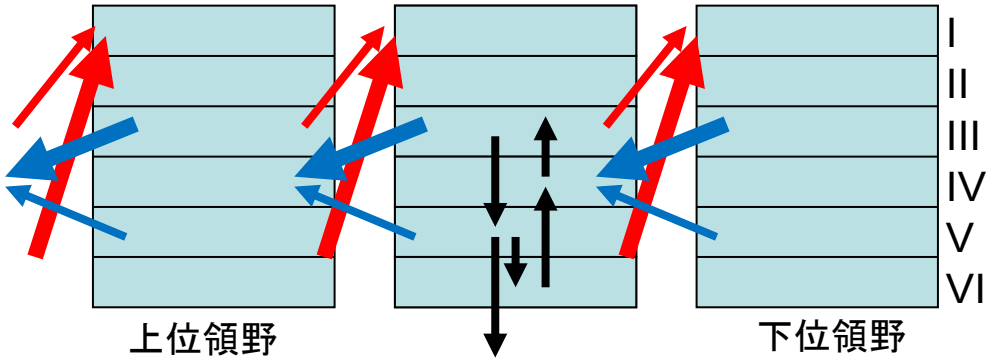
自然言語の性質を表現する道具として洗練され到達した1つの結果が単一化文法であるという事実は、脳の言語野で何らかの形で単一化が行われていることを強く示唆している。

大脳皮質に単一化が行えるなら、前頭前野も単一化を行えるはず。

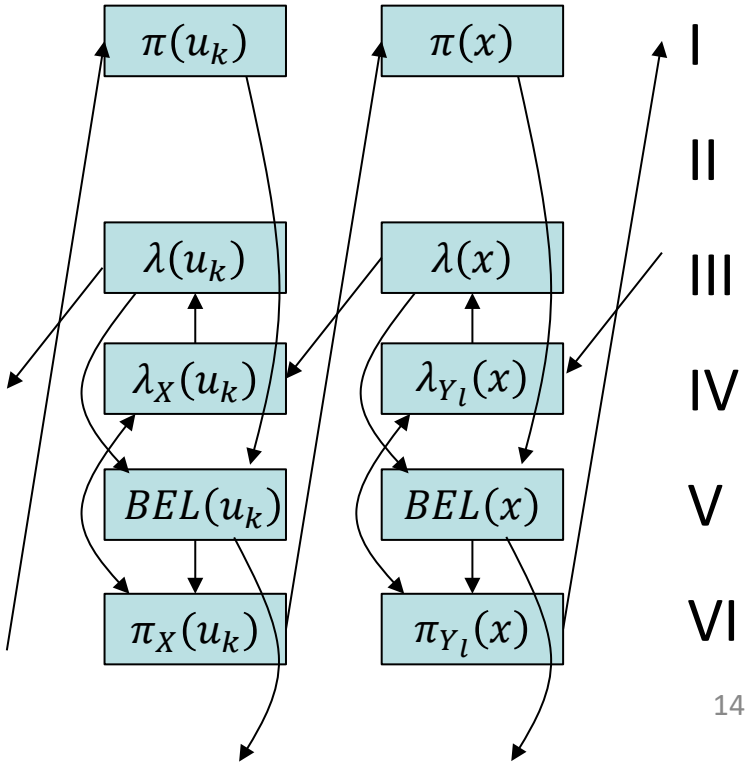
確率伝搬と大脳皮質の対応 Ver.2 [未発表]

$$BEL(x) = \alpha\lambda(x)\pi(x)$$
$$\pi(x) = \Sigma_{u_1,\ldots,u_m} P(x|u_1,\ldots,u_m)\Pi_k \pi_{X(u_k)}$$
$$\lambda(x) = \Pi_l \lambda_{Y_l}(x)$$
$$\pi_{Y_l}(x) = BEL(x)/\lambda_{Y_l}(x)$$
$$\lambda_X(u_k) = \Sigma_x \lambda(x)\Sigma_{u_1,\ldots,u_m \setminus u_k} P(x|u_1,\ldots,u_m)\Pi_{i\neq k}\pi_{X(u_i)}$$

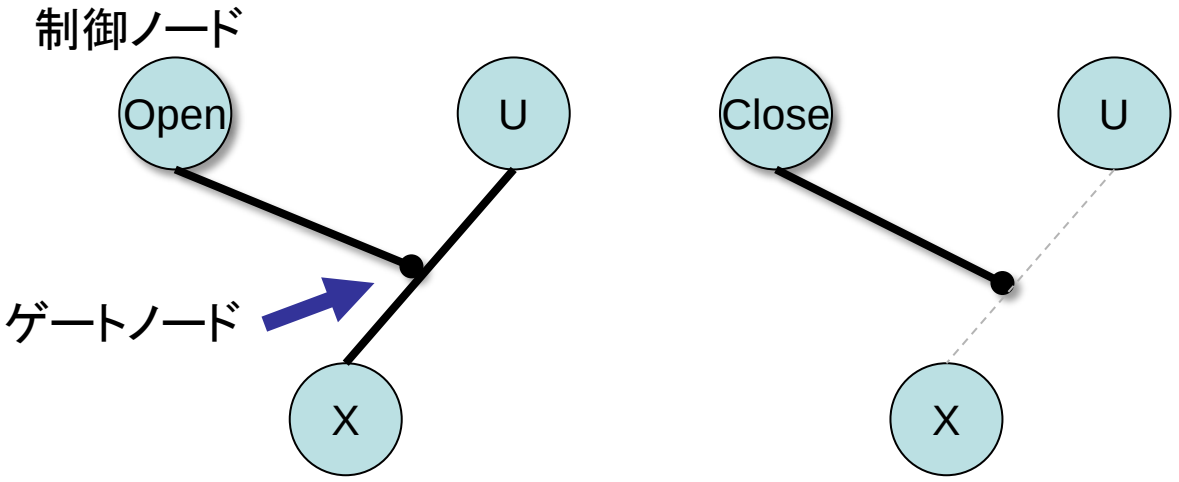




解剖学的構造



ゲートノード [一杉 2016 汎用人工知能研究会]



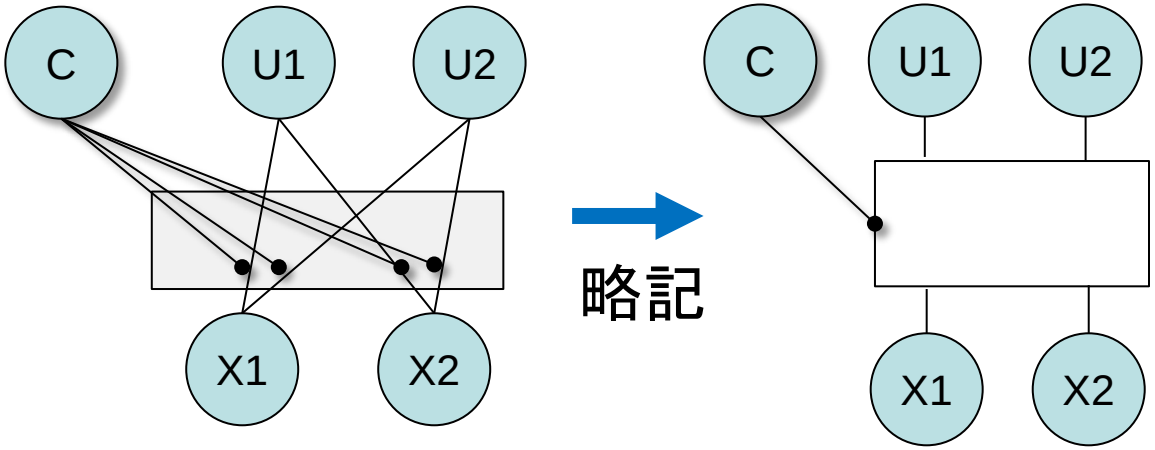
制御ノードの値が
Open なら U と X が結合、
Close なら切断

ゲートノードの条件付確率表

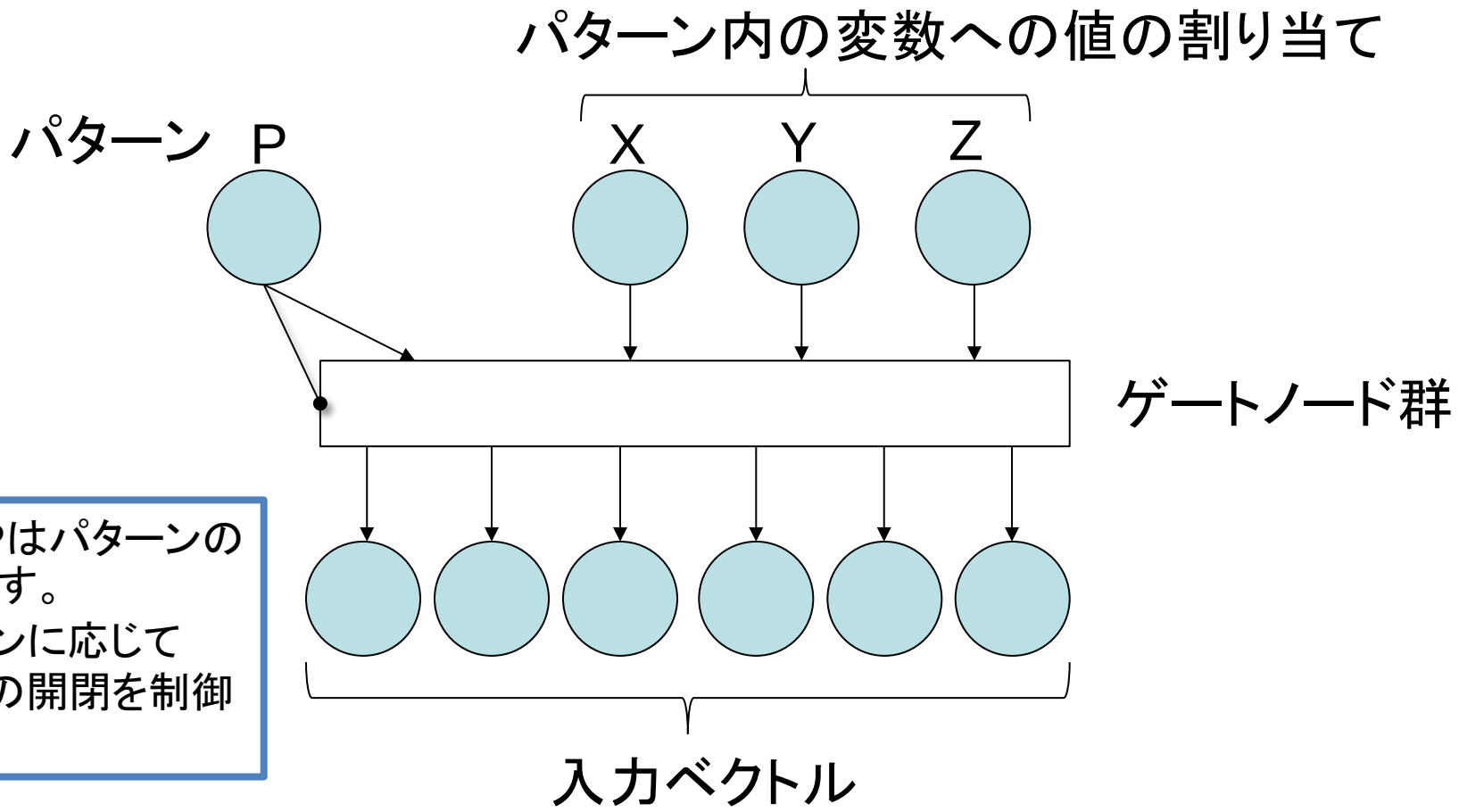
$$P(G=0|c,u) = 1-(1-w_u)^{\neg u}(1-w_c)^{c}$$
$$P(G=1|c,u) = (1-w_u)^{\neg u}(1-w_c)^{c}$$

- **論理回路のような感覚**で生成モデルを設計できる。
- **あくまでベイジアンネット**なので、出力（下流の値）から入力（上流の値）の推定もできる。

ゲートノードの行列

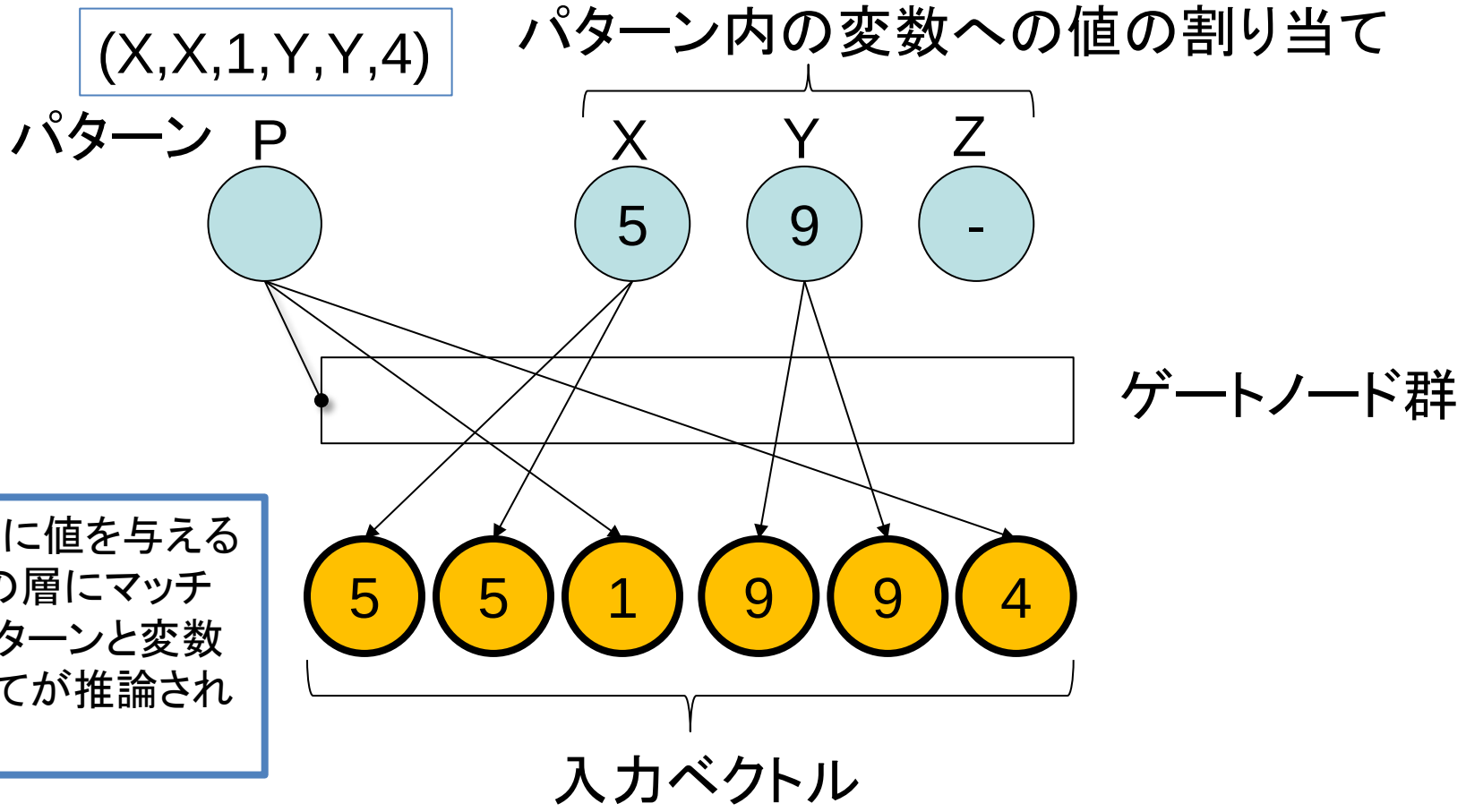


ベイジアンネットによるパターンマッチを行う回路の1つの案



ノードPはパターンのIDを表す。
パターンに応じてゲートの開閉を制御する。

パターンマッチの結果の例



下の層に値を与えると、上の層にマッチするパターンと変数割り当てが推論される。

脳におけるルール（パターン）の数

- 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憶個とし、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。

ルールの「記述実験」

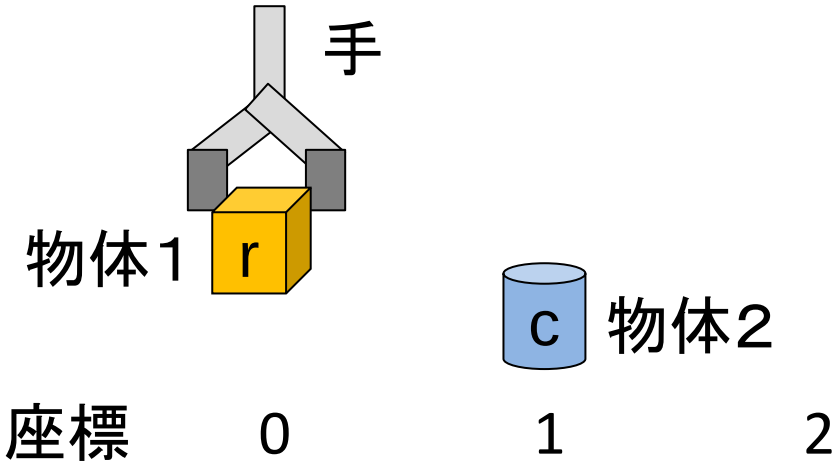
- ベイジアンネットによるパターンマッチの学習を成功させるのはおそらく簡単ではなく、**適切な目的関数の設計**が必要

- まずは理想的に圧縮されたテーブルを手で書いてみる**記述実験**により、適切な目的関数設計のヒントを得るのが今回の目的

汎用人工知能とプログラム合成

- 強化学習によるプログラム合成は汎用人工知能実現への有望なアプローチ
  - AIXI [Hutter 2000], MagicHaskeller [Katayama 2008, 2018], DNC(Differentiable Neural Computers) [Graves et al. 2016]
- どういう「プログラミング言語」を使うかが、性能に大きく影響
- RGoal のテーブルをプログラムと見なすと、それはどのような性質を持つのか？
  – それを知るのが記述実験のもう1つの目的

例題：単純化した積み木の世界



手の状態： H∈ {open, hld1, hld2}
物体１、物体２の形： O1,O2∈ {r, c}
手、物体の座標： P0,P1,P2∈ {0,1,2}
環境の状態 = (H, P0, O1, P1, O2, P2)

Cf. オブジェクトファイル
[Kahneman et al. 1992]

エージェントの動作

- 現在の状態 s 、サブゴール g のとき、[s,g,A] とマッチするルールを1つ選択。
- 単一化の結果得られた行動 A を実行する。
  - プリミティブならそれを実行。
  - サブルーチン呼び出し g(m) なら現在のサブゴール g をスタックに積み、サブゴールを m に設定。
- これを繰り返す。

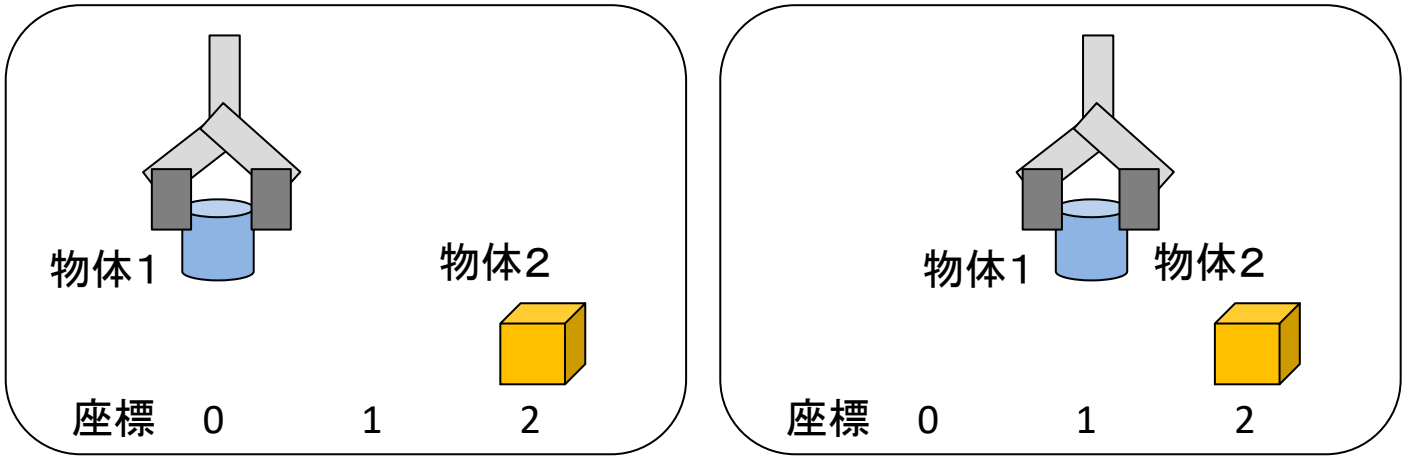
注：前回発表[一杉 2018年8月 汎用人工知能研究会] の RGoal にはスタックがないが、今回はスタックを導入

prolog 言語で「インタープリタ」を書いて記述したプログラムの動作を確認

ルールの例

s　　g　　a

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

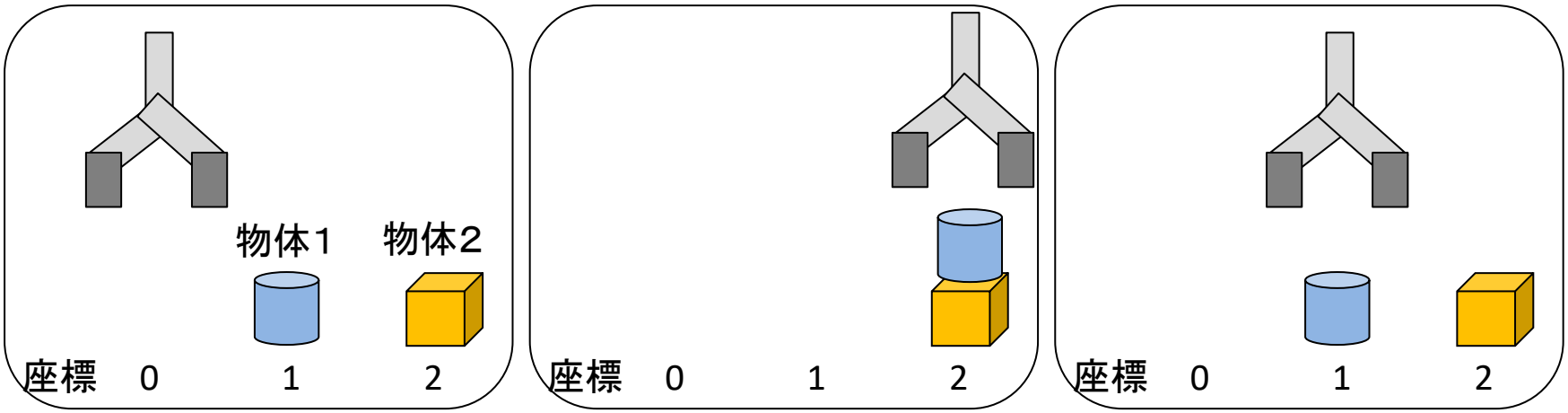


座標0で物体1をつかんだ状態から
座標1で物体1をつかんだ状態に移るためには
手を右に動かせばよい。

サブルーチンを呼び出すルールの例

s　　g　　a

m([open,**P0**,**O1**,**P1**,**O2**,**P2**], [open,**P2**,**O1**,**P2**,**O2**,**P2**], g([open,**P1**,**O1**,**P1**,**O2**,**P2**])



物体1を座標2に移動するためには、
まず手を座標1に移動させればよい。

プリミティブ行動を実行するルール

基本的な手の動かし方のルール。
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。

```
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).
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).
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).
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).
m([hld2,0,O1,P1,O2,0], [hld2,1,O1,P1,O2,1], right).
m([hld2,0,O1,P1,O2,0], [hld2,2,O1,P1,O2,2], right).
m([hld2,1,O1,P1,O2,1], [hld2,0,O1,P1,O2,0], left).
m([hld2,1,O1,P1,O2,1], [hld2,2,O1,P1,O2,2], right).
m([hld2,2,O1,P1,O2,2], [hld2,1,O1,P1,O2,1], left).
m([hld2,2,O1,P1,O2,2], [hld2,0,O1,P1,O2,0], left).
```

ルールの書き方：

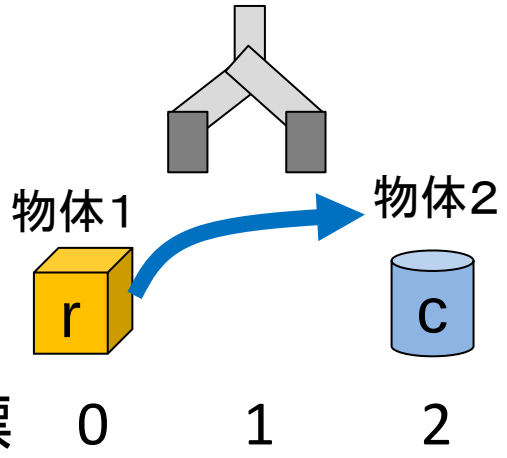
s から g に a を経由して
到達可能であれば
　　m(s,g,a).
と書く。

今回は価値は省略。

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

タスク1

物体1を、物体2がある場所に移動するタスク

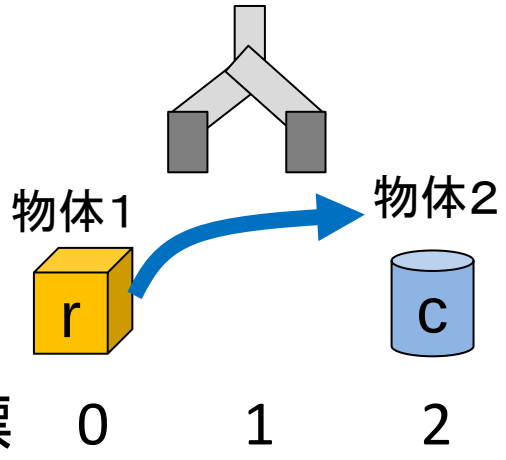


タスク1を解くプログラム：

```
% 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。
```

g([...]) という行動は、サブルーチン呼び出し

実行結果



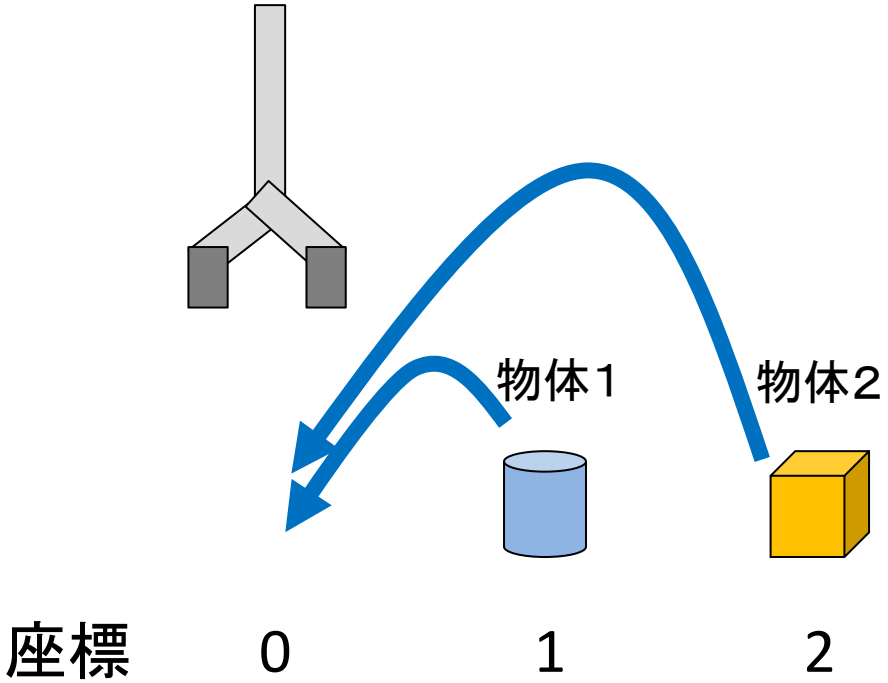
物体1を座標 0 から 2 に移動する。

実行結果（各ステップにおける (s, g, a) の組）:

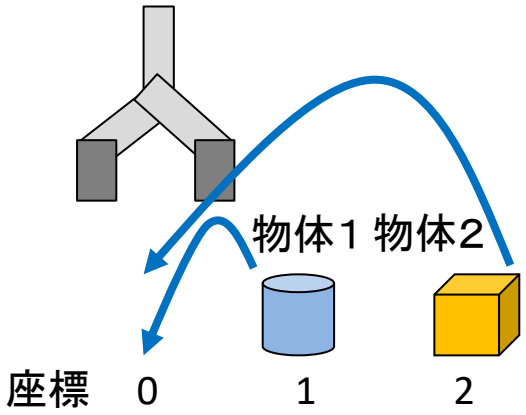
```
?- episode([open,1,r,0,c,2], [open,2,r,2,c,2],Log), printList(Log).

[[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]
```

タスク2

- 2つの物体を同じ場所に移動する。どちらを先に移動してもよい。



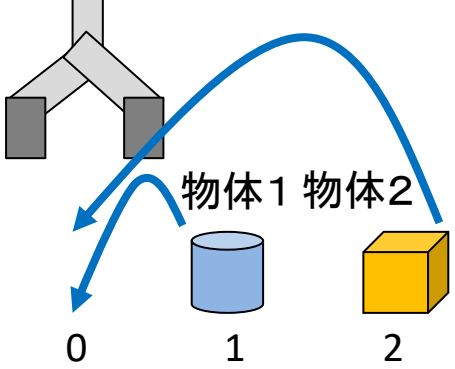
タスク2を解くプログラム




```
% 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])).


% 物体 O1 を場所 P0 に移動するサブルーチン。
m([open,P0,O1,P1,O2,P2], [open,P0,O1,P0,O2,P2], g([open,P1,O1,P1,O2,P2])).
m([open,P1,O1,P1,O2,P2], [open,P0,O1,P0,O2,P2], g([hld1,P1,O1,P1,O2,P2])).
m([hld1,P1,O1,P1,O2,P2], [open,P0,O1,P0,O2,P2], g([hld1,P0,O1,P0,O2,P2])).

% 物体 O2 を場所 P0 に移動するサブルーチン。
m([open,P0,O1,P1,O2,P2],[open,P0,O1,P1,O2,P0], g([open,P2,O1,P1,O2,P2])).
m([open,P2,O1,P1,O2,P2],[open,P0,O1,P1,O2,P0], g([hld2,P2,O1,P1,O2,P2])).
m([hld2,P2,O1,P1,O2,P2],[open,P0,O1,P1,O2,P0], g([hld2,P0,O1,P1,O2,P0])).
```

サブルーチン呼び出しがネストし、スタックが必要になる例

実行結果




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

解１：物体１、物体２の順に移動

```
[[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]
```

解２：物体２、物体１の順に移動

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

どのくらい圧縮されたのか？

- 下記パターン1つで O1,O2,P2 のすべての値の組み合わせ 2 x 2 x 3 = 12 通りを表現。

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

- 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。

手の状態： H∈ {open, hld1, hld2}
物体1、物体2の形： O1,O2∈ {r, c}
手、物体の座標： P0,P1,P2∈ {0,1,2}
環境の状態 = (H, P0, O1, P1, O2, P2)

精度の低下について

- プリミティブ行動のルールに関しては、精度の低下はない。
- タスク1、タスク2を解く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。
  - 学習速度と精度のトレードオフ。

プログラミング言語としての課題

- 再利用性のためにスタックが必要だった。
- さらに再利用性（汎化性能）を上げるために必要な言語機能：
  - 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
  - データ抽象
  - サブルーチンの終了条件

まとめ

- 単一化を用いた行動価値関数のテーブル圧縮手法を提案
  - 脳での実現方法の1つの案を提示
  - 「記述実験」により圧縮効果を予備的評価
  - RGoal をプログラミング合成システムと見なし、その特徴について考察
- 今後:
  - RGoal アーキテクチャをさらに拡張し、説得力のあるデモを動かす。