階層型強化学習 RGoal を用いた記号推論の実現手法の検討

汎用人工知能研究会

2019-08-30

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

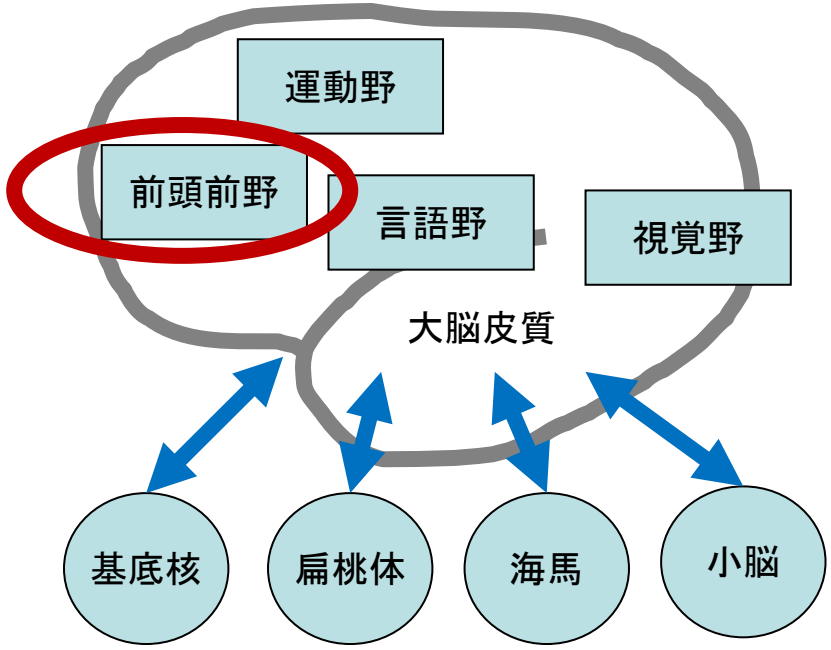
[1] National Institute of Advanced Industrial Science and Technology (AIST), AIRC

[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

私の研究の中期的目標

- 前頭前野周辺の計算論的モデルの構築
  - 報酬期待値を最大化する合理的な行動・思考・言語理解・発話をする知的エージェントのモデル



今回の目標：幼児が行うような簡単な推論の実現

「何か食べるものはないか」
「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」
「まだチョコレートが残っていて食べられるはず」

https://www.irasutoya.com/2016/12/blog-post_40.html

記号AIが扱う知識の利点

- **知識の汎用性が高い。**
  - 知識を組み合わせることで多様な推論が行える。
- **汎化能力が高い。**
- **知識のモジュラリティが高い。**
  - 環境の変化に対応しやすい。
- **自然言語との相互変換が容易。**
  - 他者との情報交換を前提とした知識の内部表現に適している。

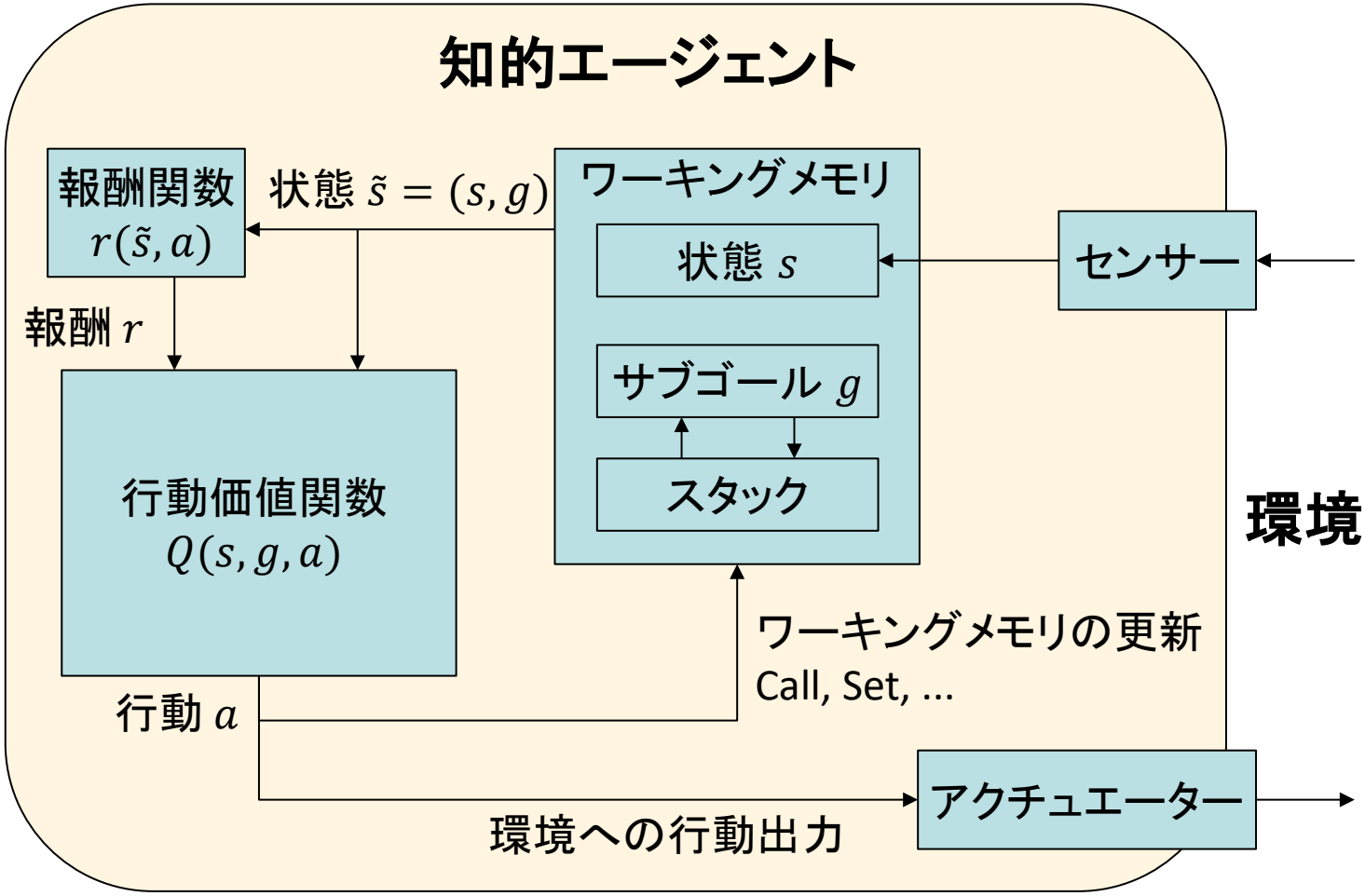
記号AIと強化学習の融合の試み

- RRL (Relational Reinforcement Learning, 2001)
  - **高い汎化能力の実現が目的**
    - 強化学習と帰納論理プログラミングの融合
    - 行動価値関数を論理決定木に圧縮
- DNC(Differentiable Neural Computers, 2016) DeepMind
  - **汎用人工知能の実現が目的**
    - NN でプログラムを表現、強化学習等で学習
- rlCoP (2018) NeurIPS
  - **定理証明システムの性能向上が目的**
    - 証明の1ステップをモンテカルロ木探索で選択
- 本研究：**ヒトと同様の推論・知識獲得の再現が目的**

アーキテクチャの説明

RGoal アーキテクチャ

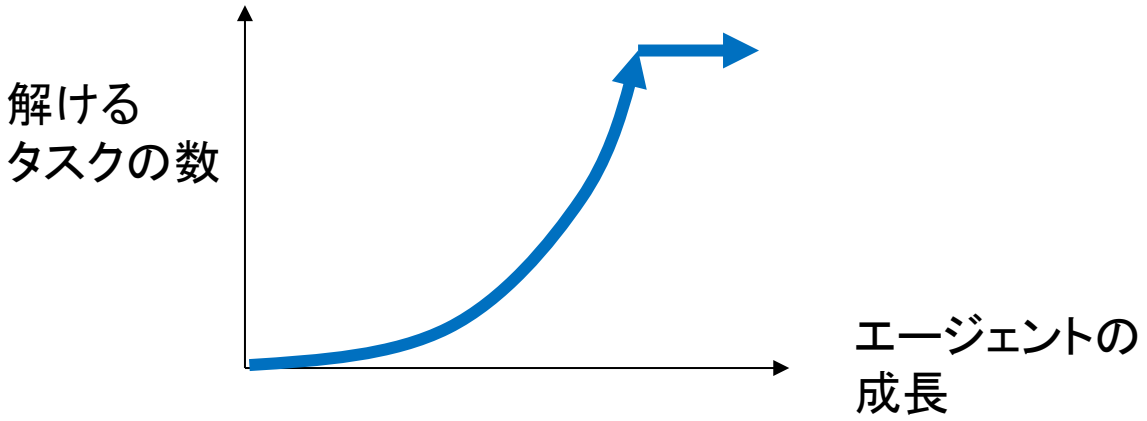
[一杉 et al.第9回 汎用人工知能研究会 2018]



サブルーチンの共有により学習を加速する階層型強化学習アーキテクチャ

なぜAGIにサブルーチンが必要か？

- **適切な教育カリキュラムがある間**は、エージェントは1つのサブルーチンを定数時間で獲得できる。
- 獲得したサブルーチンの数が増えれば、解けるタスクの数が組み合わせ**爆発的**に増える。



パターンを用いたテーブル検索

[一杉 et al.第10回 汎用人工知能研究会 2018]

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

テーブルのインデックスの代わりにパターンを用いることで、
サイズ 5 x 5 = 25 のテーブルがサイズ 4 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。

ベイジアンネットを用いた パターンマッチの回路の1つの案

(X,X,1,Y,Y,4)

パターン内の変数への値の割り当て

パターン P

X Y Z

5 9 -

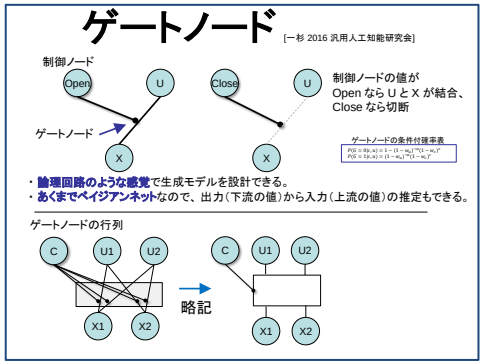
パターンに応じて
ゲートの開閉を制御

5 5 1 9 9 4

入力ベクトル

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の値が推論される。

ゲートノード群



定理証明の実現の方針

- 「真であると仮定する命題」が与えられている。
- Prolog のように、後ろ向き連鎖で証明。
  - ゴールとなる命題の証明に必要な命題を、再帰的に証明していく。
- モンテカルロでランダムに証明を探索する。
- 証明の1ステップを、強化学習における行動1ステップに対応付ける。

命題のベクトル表現

- 命題を固定長のベクトルで表現。
- 今回の実装では 以下の形の命題のみ。
  - 述語の引数は高々2つ: P(a1,a2)
  - 命題の形はＡ、A→B 、A∧B→C いずれか
- すべての命題を長さ9のベクトルで表現。
  例:

P(a,b) ∧ Q(c) → R　　: (P a b Q c 0 R 0 0)
P(a,b) → Q(c)　　: (P a b 0 0 0 Q c 0)
P(a,b)　　: (0 0 0 0 0 0 P a b)

"0" は空を表す記号

状態

- 行動価値関数 Q(s,g,a) の s, g, はそれぞれ、命題（長さ9のベクトル）で表現する。

行動

- Q(s,g,a) の a は Call(m), Set(m), Fail のいずれかとする。（今回は環境に対する行動はない。）
  - **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。
    - 行動の選択肢が他になくなった時に実行される。

行動価値関数のテーブル

RGoal では行動ルールは Q(s,g,a) の形をしたパターンで記述。

命題 A が真であると仮定されているとき:

Q((_, _ ,_), (O,O,A), Set(O,O,A)). % 命題Aは無条件に証明完了とする。

命題 A → B が真であると仮定されているとき:

Q((_,_,_), (O,O,B), Call(O,O,A)). % Aを証明する。
Q((O,O,A), (O,O,B), Set(O,O,B)). % Aが証明済みならBは証明完了とする。

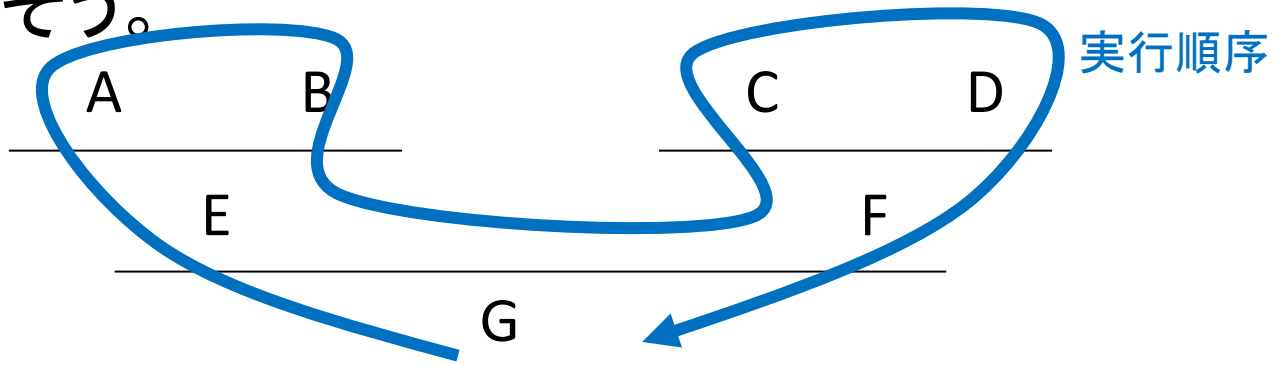
命題 A ∧ B → C が真であると仮定されているとき:

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)). % A, Bが証明済みならばCは証明完了とする。
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)).

注: "O" は空、"_" はワイルドカードを表す。

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に埋め込んで再帰呼び出しできるが、このような強力すぎる機構が脳にあるとは考えにくい。
- 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。
  - 非常に強い制限だが、それなりにいろいろなことができそう。



実行アルゴリズムの疑似コード

```
1:  procedure EPISODE(s, g)
2:      stack ← empty
3:      Choose a rule which matches Q(s, g, a) 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 Q(s', g', a') and get a' from it.
23:         s ← s'; g ← g'; a ← a'
24:     return
```

今回の追加部分

実行例

例題タスク

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。
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。
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？
Yes, X＝チョコレート

prolog による記述：

```
exist(yesterday, snack).
exist(yesterday, chocolate).
notEat(brother, chocolate).
exist(today,X) :- exist(yesterday,X), notEat(brother,X).
canEat(X) :- exist(today, X).

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

例題タスクを実行する行動ルールを Q(s,g,a) の形で表現

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))

実行ログ

```
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)
                                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)
                                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)
        Return
```

非決定論的な選択：
昨日スナックがあった。

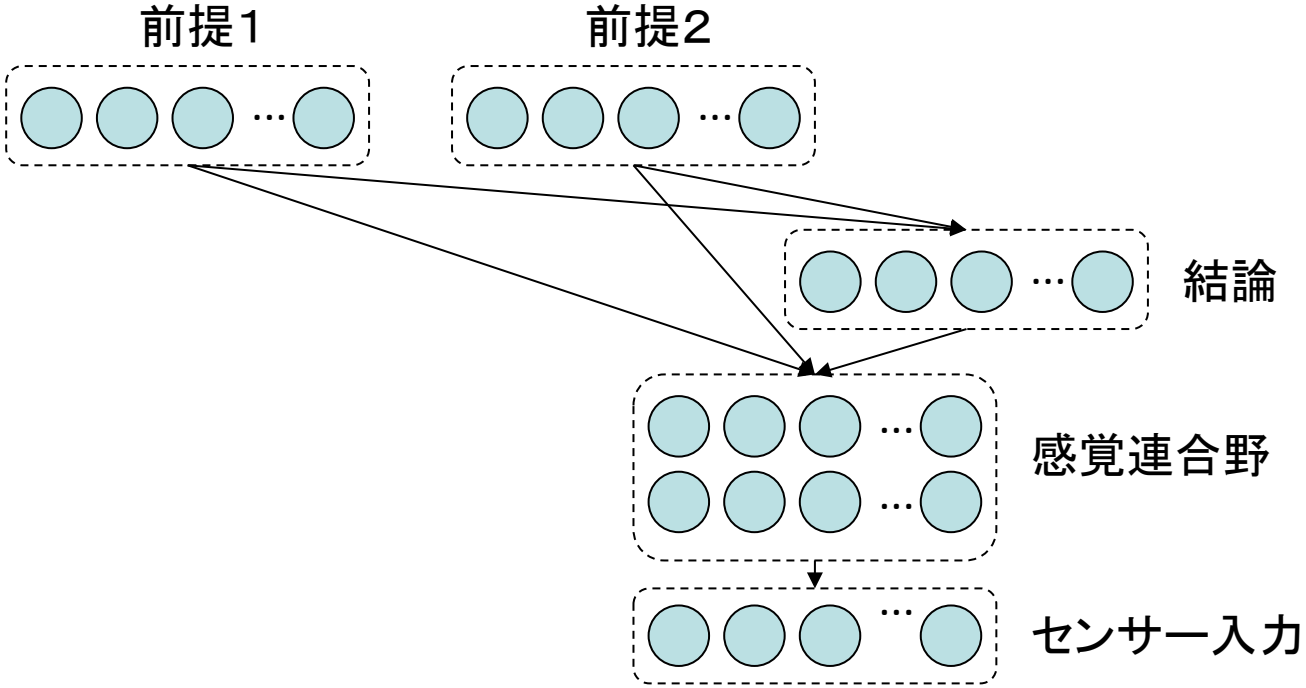
証明失敗。最初からやりなおし。

非決定論的な選択：
昨日チョコレートがあった。

証明成功。「チョコレートが食べられる」

記号的知識の獲得方法の案

- 命題は環境の状態を抽象化したもの。
- 下記のようなグラフィカルモデルで
  前提１∧前提２→結論 という命題が獲得されるのでは？



まとめ

- 強化学習の枠組みの中で定理証明器を実装。
  - RGoal に2つのアクション Set, Fail を追加。
  - パターンによるテーブル圧縮。
  - 命題を固定長のベクトルで表現。
- アルゴリズムとデータ構造は脳内の神経回路で実現可能なほどシンプル。
- 記号的知識の獲得方法について考察した。
- 自律的な推論・自律的な知識獲得の実現に近づいた。

予備スライド

記号AIと強化学習の統合の利点

- 知識を経験から獲得できる。
- 推論が学習によって効率化する。
- 矛盾のある知識も許容するような柔軟な推論システムが自然に構築できる可能性がある。
- 記号AIと統計的機械学習（教師なし学習）の統合につながる。

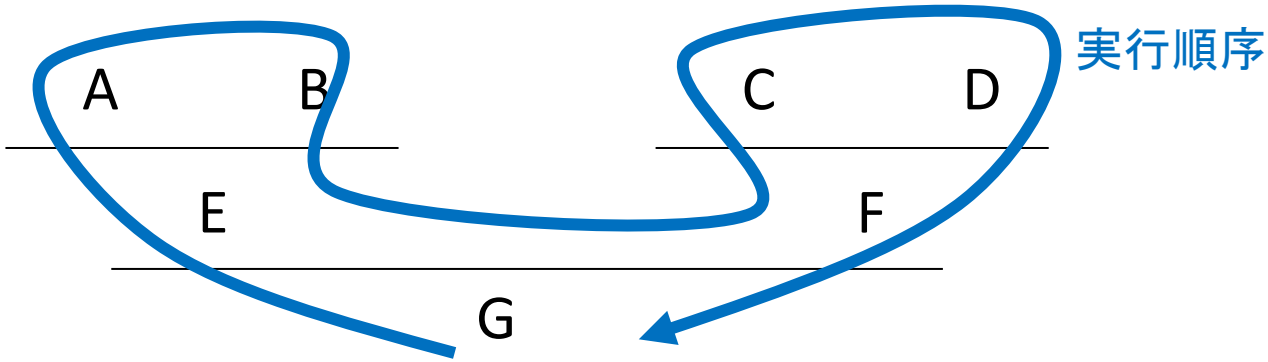
探索戦略

- 今回は単純なモンテカルロ探索。
  - 現在の s, g にマッチするルールを softmax で選択。
    - Prolog はルールを先頭から順番に試すが、本手法では価値に応じた確率でランダムに選択。
  - 選択できるルールがない場合は fail して最初からやり直す。
    - Prolog では証明の途中までバックトラックするが、本手法では最初からやり直す。

変数束縛の制限について

今回試した例題では、証明の副産物として述語の引数の値（チョコレート）が得られているが、一般にはそれができるとは限らない。

今回の実装では、変数に束縛した値が実行順序にそって伝搬していく。
prolog のような「代入の遅延」は行われない。



今後導入予定の一時メモリを使えば、この制限を超えられるかもしれない。
一時メモリには、直前の証明探索の実行ログが記録される。
もういちど証明探索を繰り返せば、直前の証明途中の情報に
アクセスできるので、実質的に未来の情報を参照できることになる。