再帰的な階層型強化学習 RGoal へのサブルーチン例外終了機能の導入

第25回 人工知能学会 汎用人工知能研究会(SIG-AGI)

一杉裕志,中田秀基,高橋直人,竹内泉（産総研）,佐野崇（東洋大）

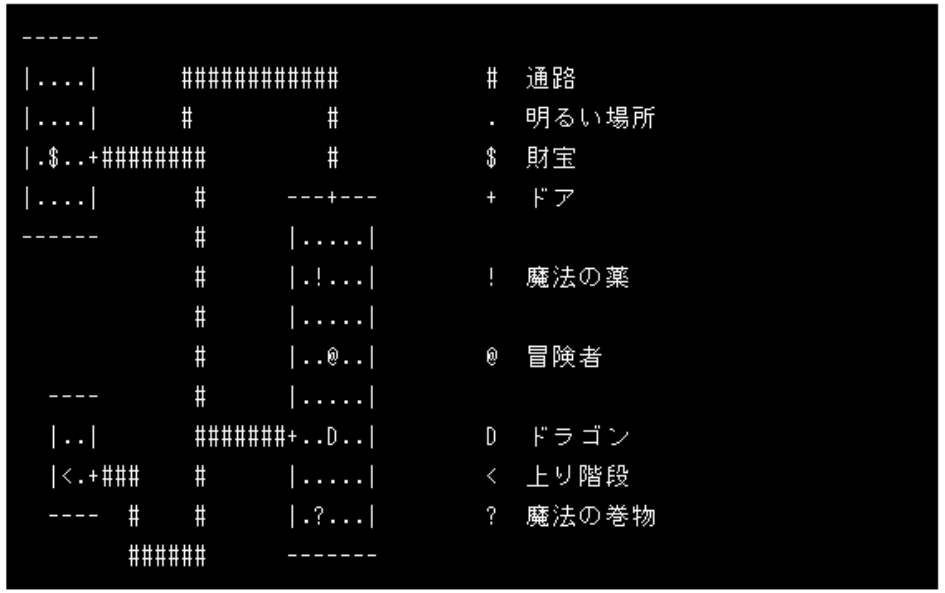
2023-11-24

私の研究の中期的目標

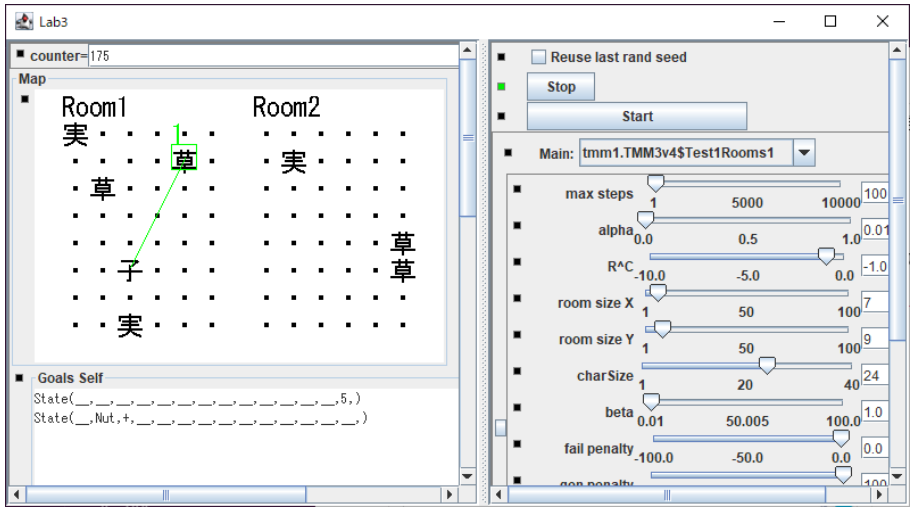
[一杉+ 第18回 汎用人工知能研究会(SIG-AGI), 2021]

- ローグライクゲームのような**実世界を極力単純化した環境で**脳型AGIのデモを動かす
- 中核技術
  - プログラム合成対象言語 Pro5Lang
    [一杉+ 第20回 汎用人工知能研究会, 2022]
  - 再帰的強化学習 RGoal
    [Ichisugi+ 2019]
  - 大脳皮質モデル BESOM
    [Ichisugi 2007]

今日の話



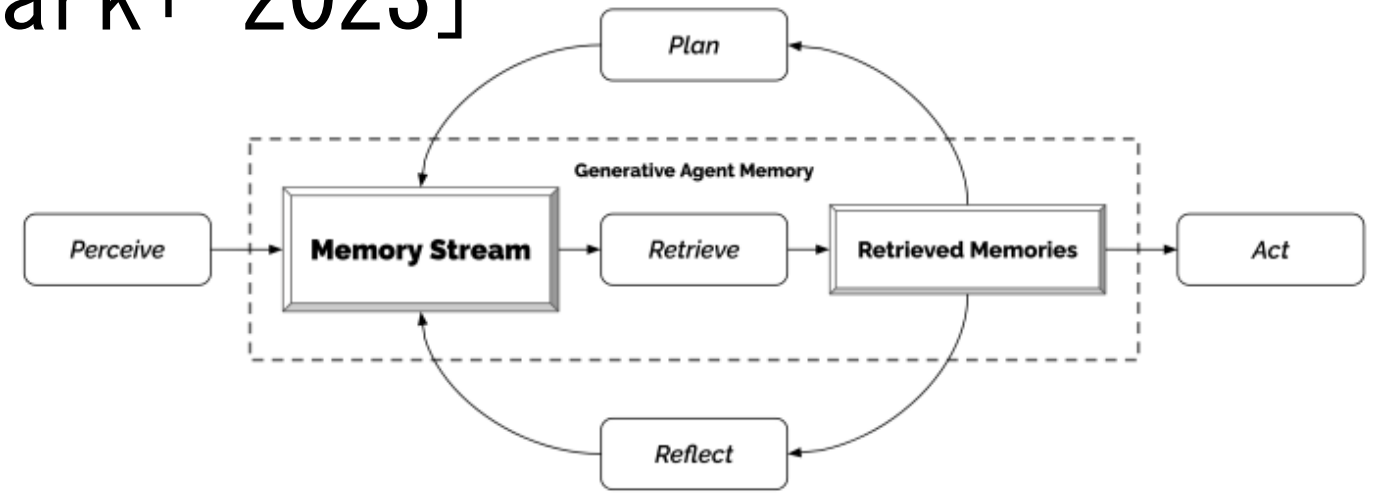
「ローグライクゲーム - Wikipedia」
https://ja.wikipedia.org/wiki/ローグライクゲーム



開発中のAGIエージェント実行環境

関連研究
Generative Agents [Park+ 2023]

- ChatGPTを使って推論・行動するAIエージェント
- テキストベースの「エピソード記憶」や「ワーキングメモリ」を持つ
- エージェントどうしが町で対話などをしながら生活
- 「動機」を人間が与えて行動をシミュレーション
  - 「バレンタインパーティーを開きたい」
    → 数日後パーティーが開かれた



https://arxiv.org/pdf/2304.03442.pdf

本発表の概要

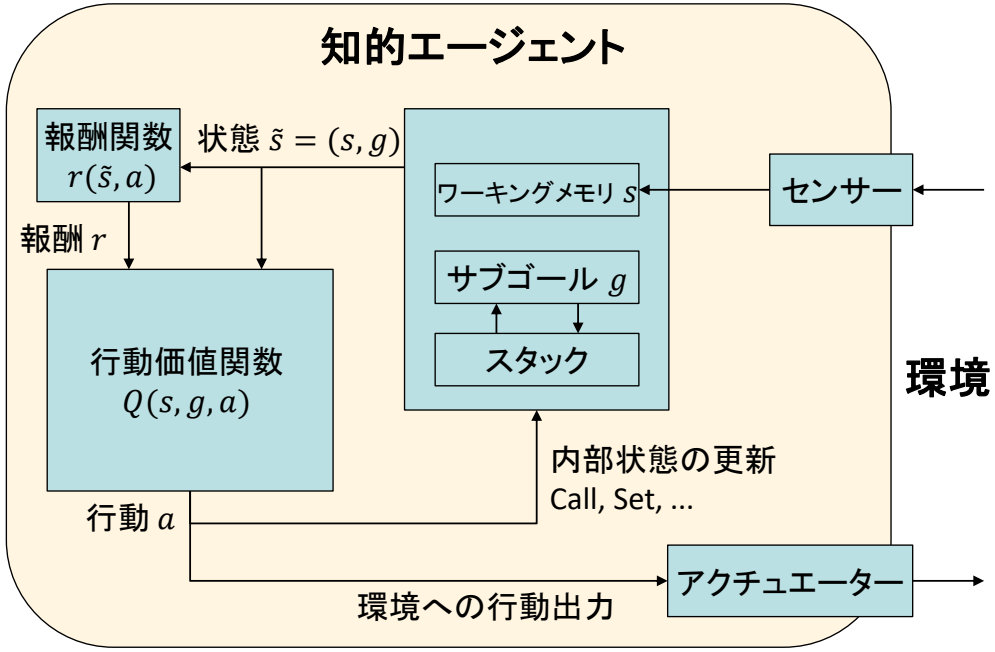
- 以前提案した再帰的強化学習 RGoal [Ichisugi+ ICANN 2017] は
  サブルーチンの出口が１つの状態のみ → 実用的ではない
- 本発表
  - サブルーチンが複数の出口を持てるように拡張
  - 例外終了の機能を追加
  - ２つの学習アルゴリズムの実装：モンテカルロ法、Sarsa
    （ただし、アルゴリズムは改良の余地あり）

階層型強化学習 RGoal [Ichisugi+ ICANN 2019]

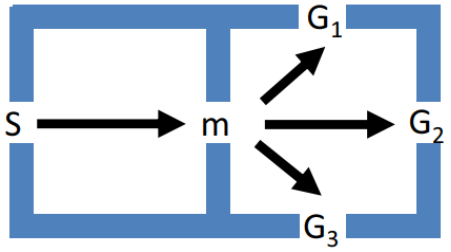
- 再帰的なサブルーチン呼び出しが可能
- Prolog 言語に似た方法（後ろ向き探索）で推論タスクも解ける [一杉+ 第12回 汎用人工知能研究会 2019]
  → 前頭前野による推論のモデルを目指す

サブルーチンの出口が1つの場合の学習則:

$Q(s,g,a) \leftarrow Q(s,g,a) + \alpha(r + Q(s',g',a') - Q(s,g,a) + V_g(g'))$

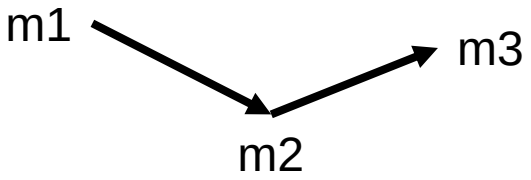


多層の階層型強化学習 MAXQ [Dietterich 2000]



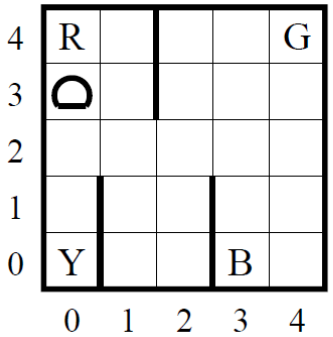
１．**サブタスク共有**：
タスク間でサブルーチンを共有することで学習を加速

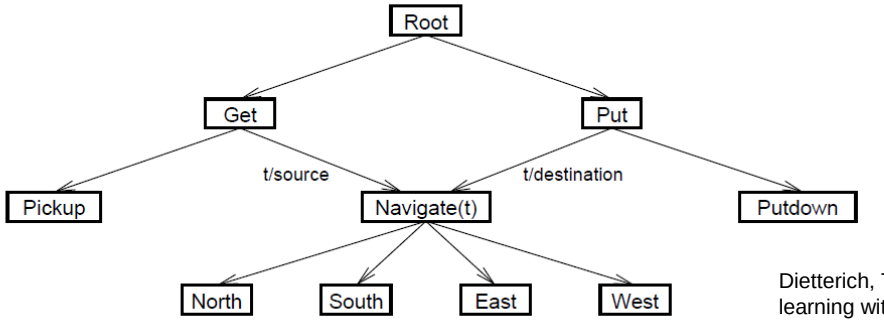
２．**時間抽象**：
経路探索をサブルーチン単位で行うことで学習を加速



３．**状態抽象**：
サブルーチンに関係ない情報を無視することで学習を加速

- 行動価値関数の計算が複雑
- 多層だが層の数は**固定**





Dietterich, T. G.: Hierarchical reinforcement learning with the MAXQ value function decomposition, Journal of artificial intelligence research, Vol. 13, pp. 227–303 (2000)

いらすとや
https://www.irasutoya.com/2012/12/blog-post_3849.html

Recursive Reinforcement Learning
[Hahn+ NeurIPS 2022]

- Recursive MDP :
  MDP でモデル化されるサブルーチンどうしが相互に再帰呼び出し
- それを解くアルゴリズムは Recursive Q-Learning は一般には複雑
- 出口が常に１つの場合は旧 RGoal と本質的に同じ

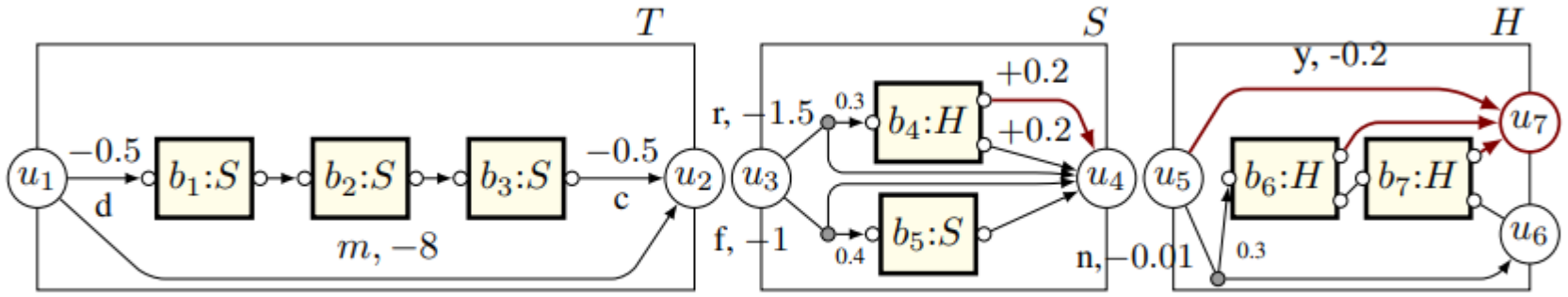


Figure 1: A recursive Markov decision process with three components T, S, and H.

Hahn, E. M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., and Wojtczak, D.: Recursive Reinforcement Learning, in Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. eds., Advances in Neural Information Processing Systems (2022)

階層型強化学習の利点の１つ：サブタスク共有

タスク1

サブルーチン g

S1 → … → … → … → … → … → … → … → G1

タスク2

サブルーチン g

S2 → … → … → … → … → … → … → G2

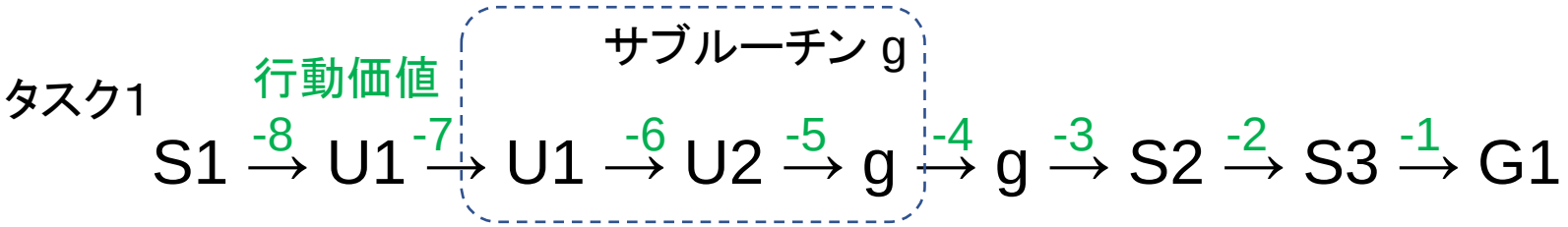
サブルーチン呼び出し文脈が異なっていても、
サブルーチン内の最適方策はほぼ同じはず

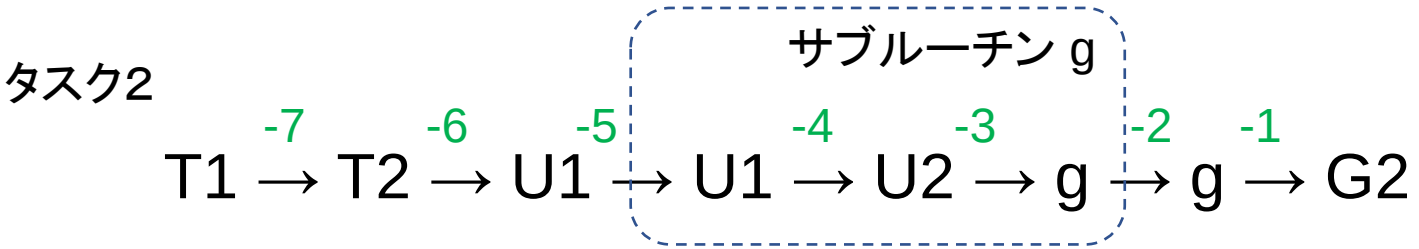
例：はしごを使って枝を切る、はしごを使って電球を替える、・・・

いらすとや
https://www.irasutoya.com/2017/04/blog-post_516.html

行動価値は文脈によって異なる

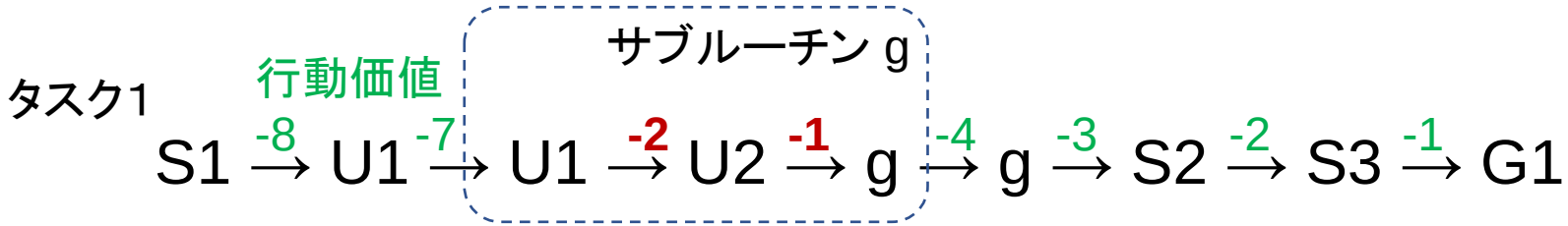
1ステップごとに -1 の負の報酬（コスト）が与えられる場合、
行動価値＝ゴール到達までのコスト

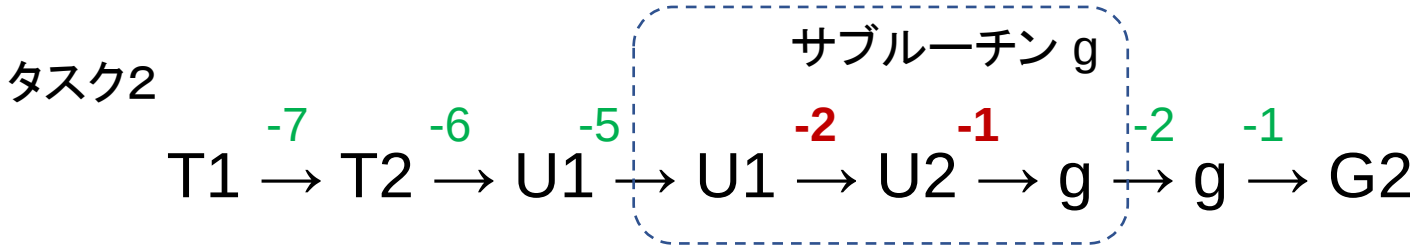




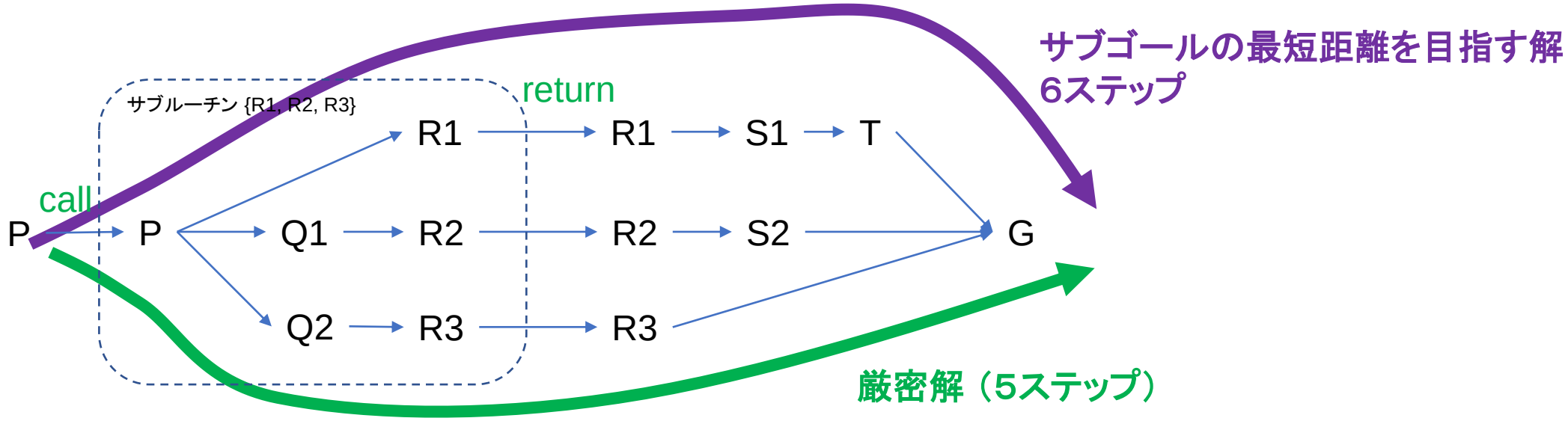
１つの解決策：
行動価値＝サブゴール到達までの累積報酬

MAXQ, 旧 RGoal など



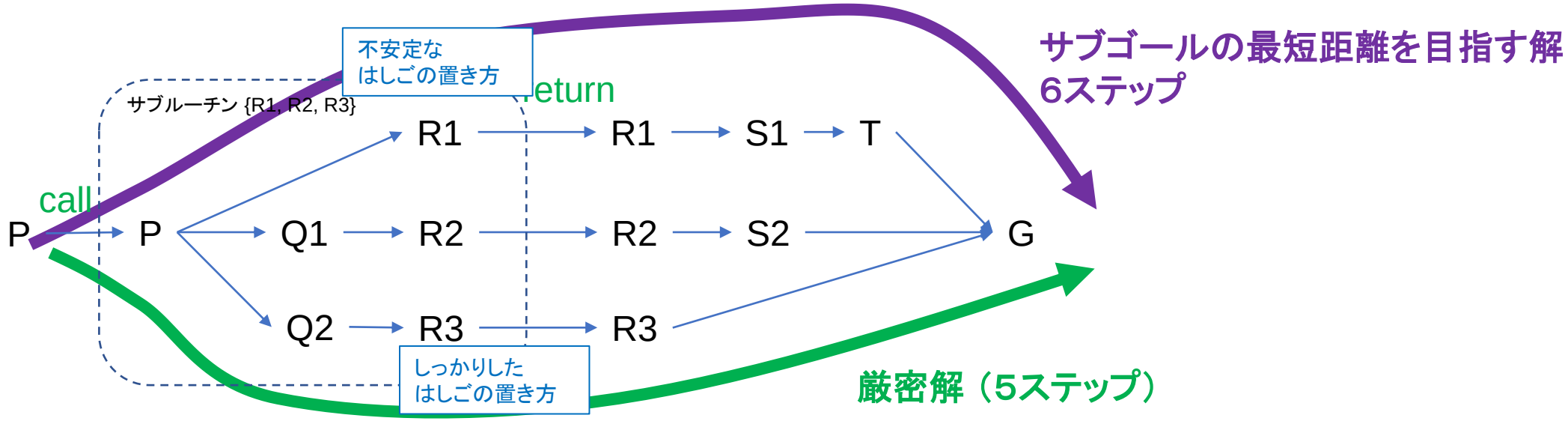


サブルーチンに複数の出口がある場合に起きる問題



文脈を無視し、単純にサブゴールへの最短距離を目指すアルゴリズムは、
サブルーチンの共有による汎化はしやすいが、
厳密解を得られない

サブルーチンに複数の出口がある場合に起きる問題



文脈を無視し、単純にサブゴールへの最短距離を目指すアルゴリズムは、
サブルーチンの共有による汎化はしやすいが、
厳密解を得られない

Recursive Q-Learning

[Hahn+ NeurIPS 2022]

出口の価値のベクトルを
文脈とみなし、
文脈ごとに価値関数を学習

→ 問題を厳密にモデル化するが
学習パラメタは多くなり、
汎化しにくくなる

Algorithm 1: Recursive Q-learning

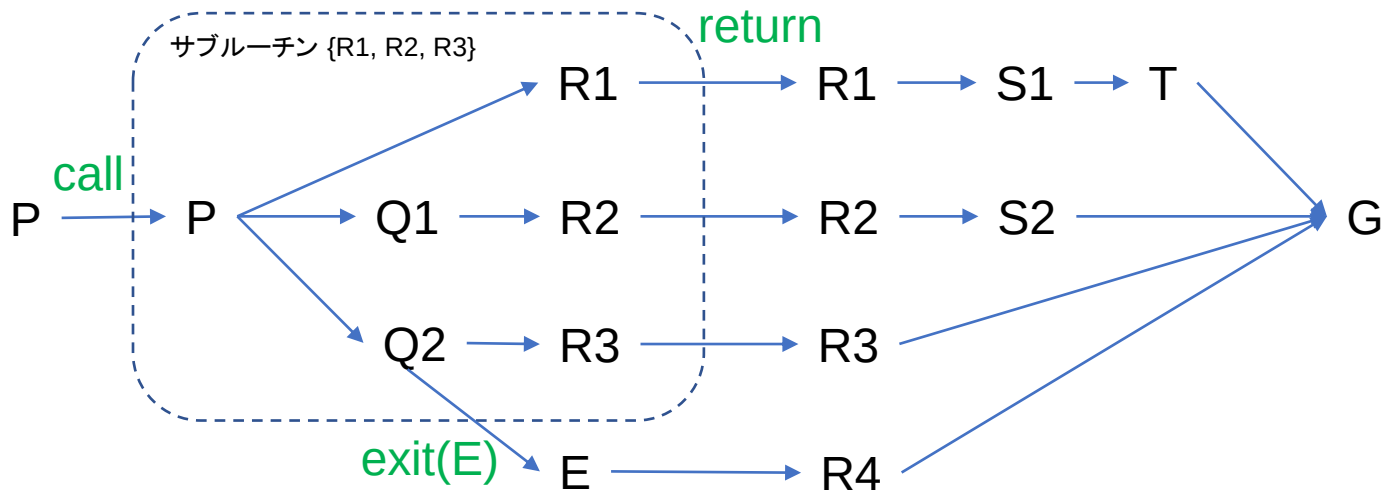
Initialize $Q(s, v, a)$ arbitrarily
while *not converged* **do**
  $v \leftarrow \mathbf{0}$
  stack $\leftarrow \emptyset$
  Sample trajectory $\tau \sim \{(s, a, r, s'), ...\}$
  for s, a, r, s' *in* τ **do**
    Update α_i according to learning rate schedule
    if *entered box* **then**
      $\{s_{\text{exit}_1}, \ldots, s_{\text{exit}_n}\} \leftarrow \text{getExits}(s')$
      $v' \leftarrow [\max_{a' \in A(s_{\text{exit}_1})} Q(s_{\text{exit}_1}, v, a'), \ldots, \max_{a' \in A(s_{\text{exit}_n})} Q(s_{\text{exit}_n}, v, a')]$
      $v'_{\min} \leftarrow \min(v')$
      $v' \leftarrow v' - v'_{\min}$
      $Q(s, v, a) \leftarrow (1 - \alpha_i) Q(s, v, a) + \alpha_i (r + \max_{a' \in A(s')} Q(s', v', a') + v'_{\min})$
      stack.push(v)
      $v \leftarrow v'$
    else if *exited box* **then**
      $\{s_{\text{exit}_1}, \ldots, s_{\text{exit}_n}\} \leftarrow \text{getExits}(s)$
      Set k such that $s' = s_{\text{exit}_k}$
      $Q(s, v, a) \leftarrow (1 - \alpha_i) Q(s, v, a) + \alpha_i (r + v(k))$
      $v \leftarrow$ stack.pop()
    else
      $Q(s, v, a) \leftarrow (1 - \alpha_i) Q(s, v, a) + \alpha_i (r + \max_{a \in A(s')} Q(s', v, a'))$
    end
  end
end
return Q

Hahn, E. M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., and Wojtczak, D.: Recursive Reinforcement Learning, in Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. eds., Advances in Neural Information Processing Systems (2022)

複数の出口と例外終了

- 正常終了
  - 推論、観測行動の結果
    例：
    「戸棚におかしがあるか確認する」というサブルーチンの出口は
    「あることを確認した」「ないことを確認した」の２つ
  - 複数の目的の達成の仕方
    例：「はしごを置く」の結果が、しっかり置けている、不安定、など
  - 副作用
    例：「おかしを食べる」の結果の、まわりのちらかり具合
- 例外終了
  - サブゴールが達成されないままサブルーチンを抜けざるを得ない場合がある
  - 例：
    探したが見つからない、食べようとしたが袋が開けられない、親に何か教えてもらおうとしたら親も知らない

複数の出口と例外終了があるサブルーチン



旧 RGoal の考え方での問題点:

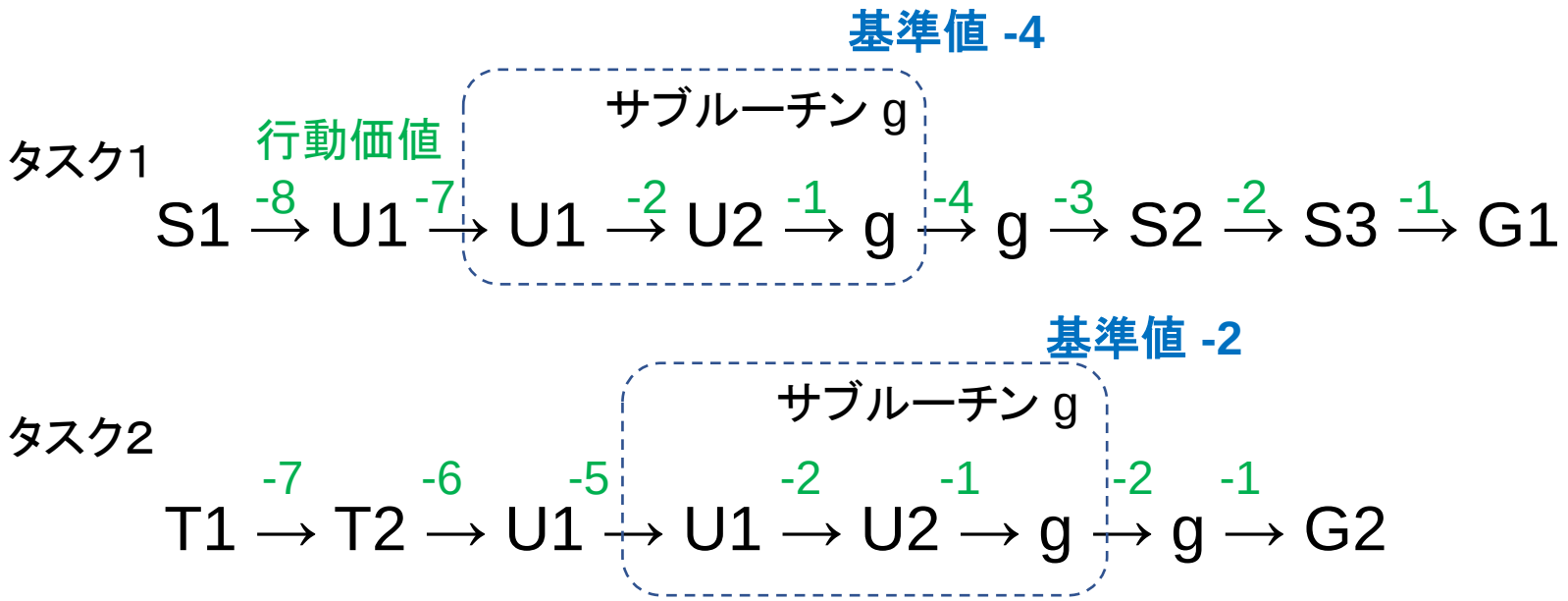
- 出口が複数あるときは価値の計算が簡単でない
- 例外的な出口があるときの対処方法が自明でない

本研究の動機のまとめ

- サブルーチンを出た後の報酬をサブルーチン内部の方策に反映させたい
  → 再帰的強化学習をプログラム合成に適用する際に必須
- 異なる文脈への汎化能力と精度の間のうまいバランスを取りたい
  → オンライン性能を上げるために必須
- サブルーチンの例外終了も扱いたい
  → リアリティのあるデモを動かす際に必須
- （脳が実行可能なほどシンプルなアルゴリズムにしたい）
  - → 前頭前野の計算論的モデルとして必須

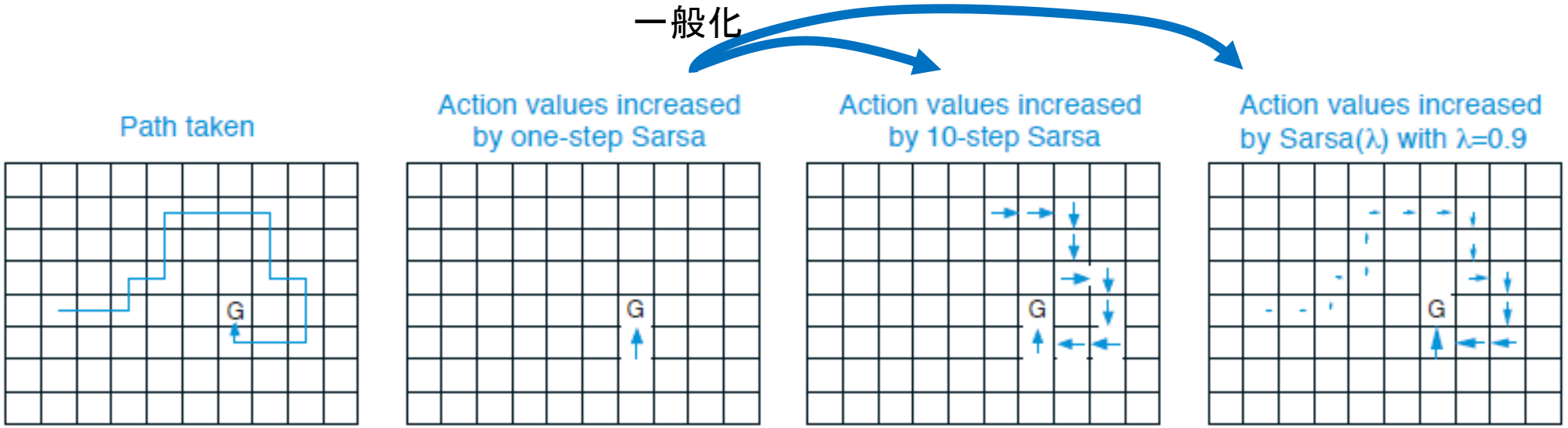
新しいアルゴリズムの設計方針

- サブルーチンを正常終了したときの状態の価値の**方策πにおける期待値**を**基準値**として、
  サブルーチン内の**相対価値**を学習する
- 基準値は、**異なる呼び出し文脈ごと**に学習する



モンテカルロ法，n-step Sarsa，Sarsa(λ)

一般化



one-step Sarsa
＝ 普通の Sarsa

n-step Sarsa

λは適格度トレースにおける適格度の減衰係数

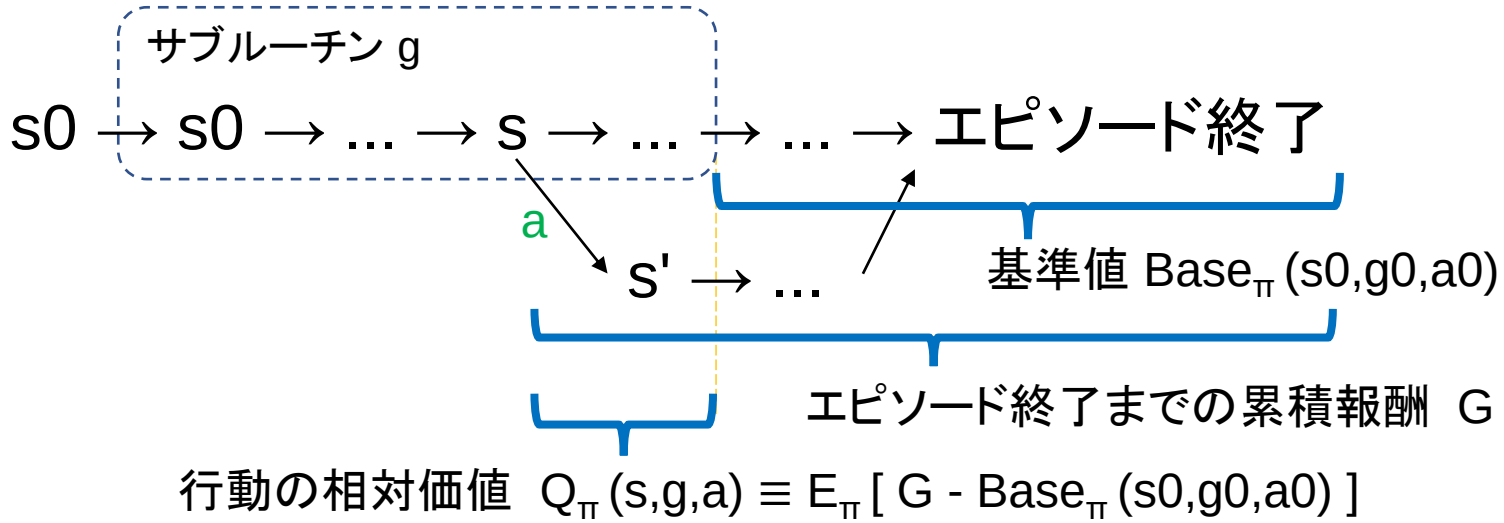
Sarsa(0)
＝ 普通の Sarsa

Sarsa(1)
≒∞-step Sarsa
≒モンテカルロ法

Reinforcement Learning: An Introduction, second edition
Richard S. Sutton and Andrew G. Barto

モンテカルロ法での学習則

注:シングルタスクの場合
マルチタスクでは修正が必要



サブルーチンを正常終了したときの基準値の学習則

$$Base(s0,g0,a0) \leftarrow Base(s0,g0,a0) + \alpha(G - Base(s0,g0,a0))$$

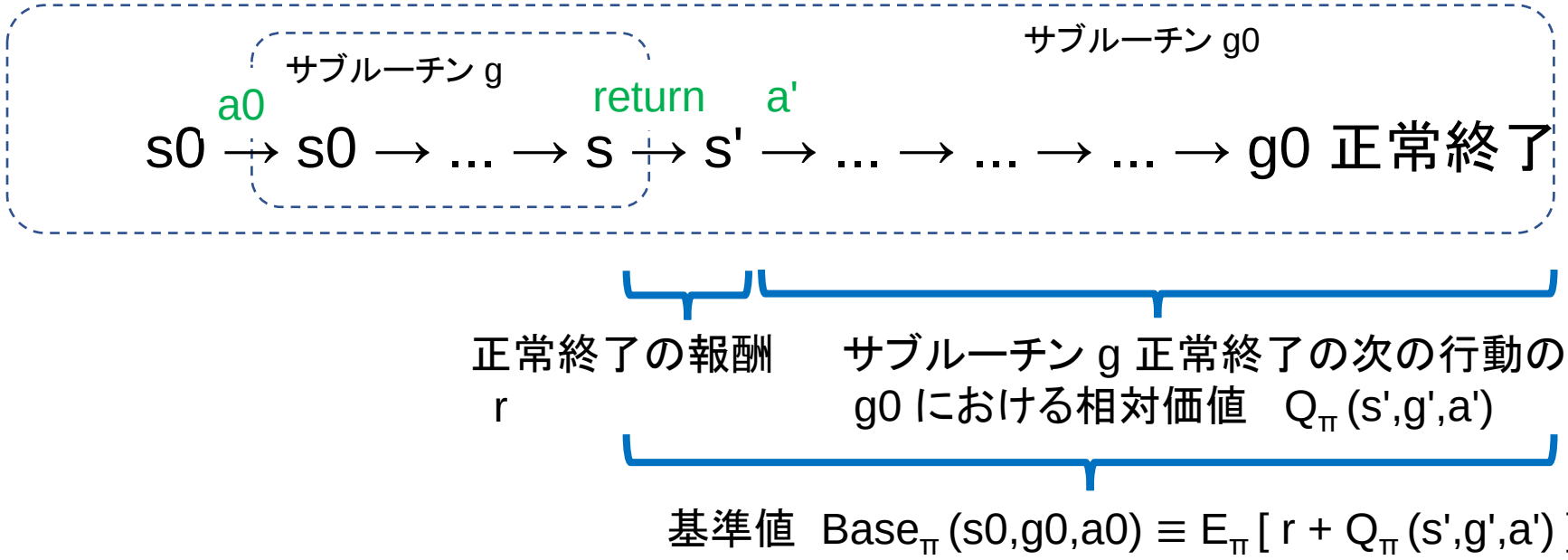
サブルーチン内部の行動価値の学習則

$$Q(s,g,a) \leftarrow Q(s,g,a) + \alpha(G - Base(s0,g0,a0) - Q(s,g,a))$$

サブルーチン内の価値基準値の学習方法（Sarsa）

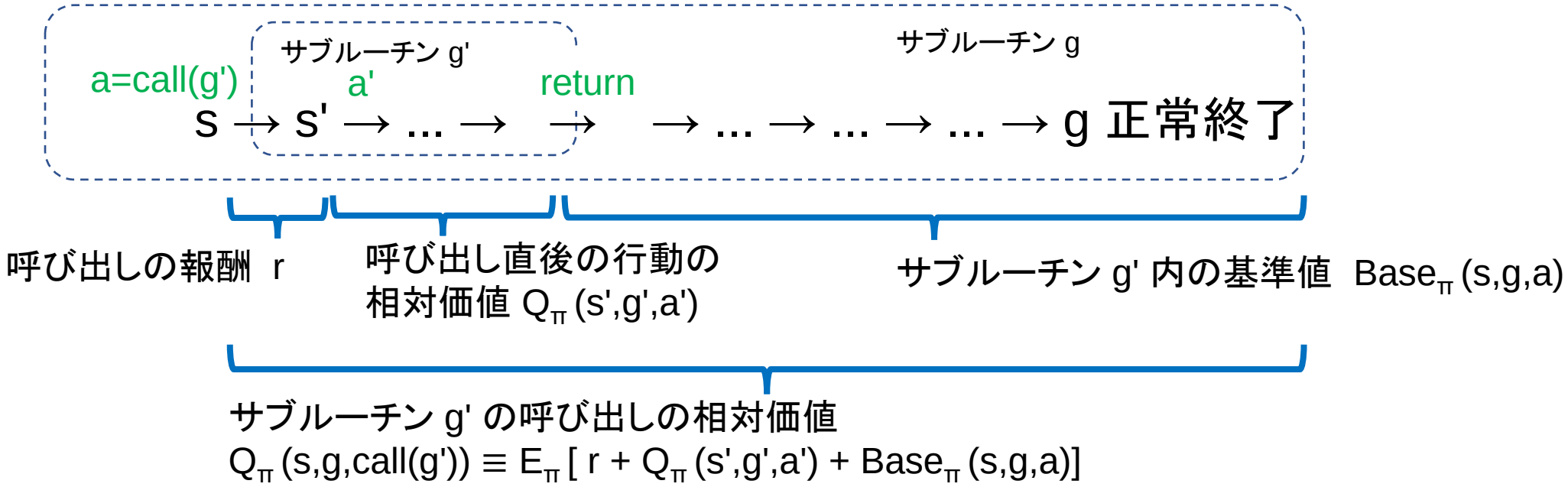
サブルーチンを正常終了したときに下記の学習則で学習

Base(s0,g0,a0) ← Base(s0,g0,a0) + α(r + Q(s',g',a') - Base(s0,g0,a0))



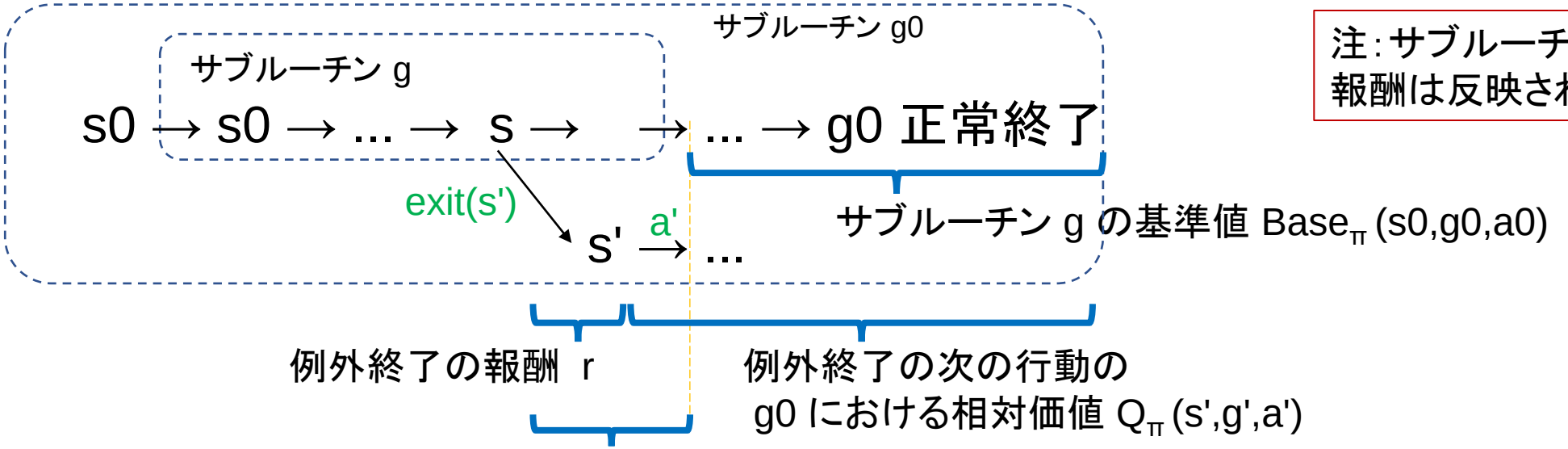
サブルーチン呼び出しの価値の学習方法（Sarsa）

$$Q(s,g,a) \leftarrow Q(s,g,a) + \alpha(r + Q(s',g',a') + Base(s,g,a) - Q(s,g,a))$$



サブルーチンが例外終了した時の学習則(Sarsa)

$$Q(s,g,a) \leftarrow Q(s,g,a) + \alpha(r + Q(s',g',a') - Base(s0,g0,a0) - Q(s,g,a))$$



注:サブルーチン終了後の報酬は反映されない

疑似コード

今回はシンプルさを
特に重視

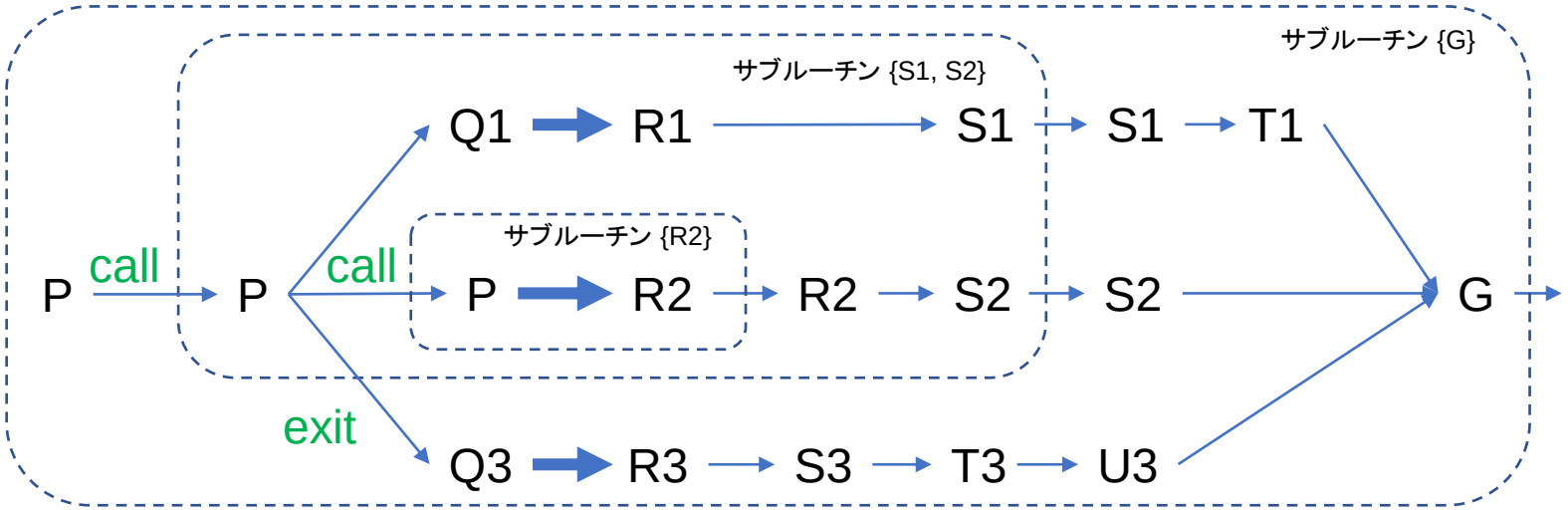
```
1: procedure EPISODE(S, G)
2:     # Init and choose the first action rule.
3:     ruleHistory, callerHistory, rewardHistory ← empty
4:     s ← S;  g ← G;  stack, callerStack ← empty
5:     stack.push(G) ;  caller ← StartRule ;  callerStack.push(dummy)
6:     Choose rule from s, g using policy derived from Q
7:     while stack is not empty do
8:         # Take action.
9:         a ← getAction(rule)
10:        if a = RET then
11:            s' ← s;  g' ← stack.pop();  reward ← R^Return
12:        else if a is exit(m) then
13:            s' ← m;  g' ← stack.pop();  reward ← R^Exit
14:        else if a is call(m) then
15:            stack.push(g);  s' ← s;  g' ← m;  reward ← R^Call
16:        else
17:            Take action a, observe reward and s';  g' ← g
18:        # Choose action rule.
19:        if s' ∈ g' then
20:            rule' ← ReturnRule
21:        else
22:            Choose rule' from s', g' using policy derived from Q
23:        # Remember history for Update.
24:        ruleHistory.push(rule) ;  rewardHistory.push(reward) ;  callerHistory.push(caller)
25:        if a = RET then
26:            caller ← callerStack.pop()
27:        else if a is exit(m) then
28:            caller ← callerStack.pop()
29:        else if a is call(m) then
30:            callerStack.push(caller)
31:            caller ← rule
32:        #
33:        s ← s';  g ← g';  rule ← rule'
34:    Update(ruleHistory, callerHistory, rewardHistory)
35:
36: procedure UPDATE(ruleHistory, callerHistory, rewardHistory)
37:    G ← 0    # Total reward until the end of the episode.
38:    while ruleHistory is not empty do
39:        rule ← ruleHistory.pop() ;  caller ← callerHistory.pop() ;  G ← G + rewardHistory.pop()
40:        if rule = ReturnRule then
41:            Base(caller) ← Base(caller) + α(G − Base(caller))
42:        else
43:            Q(rule) ← Q(rule) + α(G − Base(caller) − Q(rule))
```

図 3: モンテカルロ法による学習の疑似コード。ここで rule とは (s, g, a) の組を表すデータ構造である。ReturnRule は定数で getAction(ReturnRule)=RET, Q(ReturnRule)=0 とする。StartRule はエピソードを開始するダミーの行動ルールを表す定数。$R^{Call}, R^{Return}, R^{Exit}$ はそれぞれサブルーチン呼び出し、 RET、 exit の報酬（実行コスト）を表す定数。

```
1: procedure EPISODE(S, G)
2:     # Init and choose the first action rule.
3:     s ← S;  g ← G;  stack, callerStack ← empty
4:     stack.push(G) ;  caller ← StartRule ;  callerStack.push(dummy)
5:     Choose rule from s, g using policy derived from Q
6:     while stack is not empty do
7:         # Take action.
8:         a ← getAction(rule)
9:         if a = RET then
10:            s' ← s;  g' ← stack.pop();  reward ← R^Return
11:        else if a is exit(m) then
12:            s' ← m;  g' ← stack.pop();  reward ← R^Exit
13:        else if a is call(m) then
14:            stack.push(g);  s' ← s;  g' ← m;  reward ← R^Call
15:        else
16:            Take action a, observe reward and s';  g' ← g
17:        # Choose action rule.
18:        if s' ∈ g' then
19:            rule' ← ReturnRule
20:        else
21:            Choose rule' from s', g' using policy derived from Q
22:        # Update.
23:        if a = RET then
24:            Base(caller) ← Base(caller) + α (reward + Q(rule') − Base(caller))
25:            caller ← callerStack.pop()
26:            # Do not update Q(rule) because Q(ReturnRule) is always 0 .
27:        else if a is exit(m) then
28:            Q(rule) ← Q(rule) + α (reward + Q(rule') − Base(caller) − Q(rule))
29:            caller ← callerStack.pop()
30:        else if a is call(m) then
31:            callerStack.push(caller)
32:            caller ← rule
33:            Q(rule) ← Q(rule) + α (reward + Q(rule') + Base(rule) − Q(rule))
34:        else
35:            Q(rule) ← Q(rule) + α (reward + Q(rule') − Q(rule))
36:        #
37:        s ← s';  g ← g';  rule ← rule'
```

図 7: Sarsa による学習の疑似コード

テスト用タスク



・サブルーチンの正常終了を目指すか、
別のサブルーチンを呼び出すか、
例外終了するかを
合理的に選択することを確認するテストプログラム

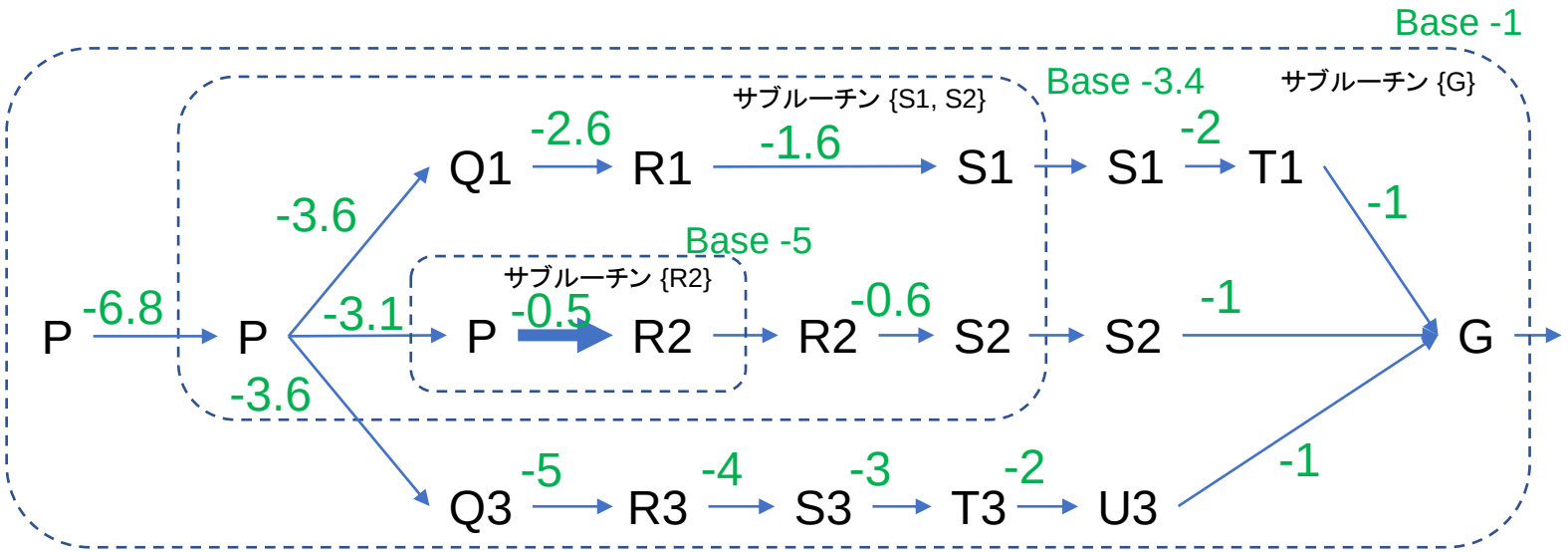
・太い矢印のコストが可変
・デフォルトではすべての状態遷移のコストは -1 とする

Pro5Lang でのソースコード

```
StateN g0 = w(a(PLS,G));
StateN g1 = w(a(PLS,S));
StateN g2 = w(a(2,R));

setRulesName("g0");
rule(w(a(1,P)), g0, call(g1));        // 1
// S1 -> T1 -> G1
rule(w(a(1,S)), g0, set(a(1,T)));     // 2
rule(w(a(1,T)), g0, set(a(1,G)));     // 3
// S2 -> G1
rule(w(a(2,S)), g0, set(a(1,G)));     // 4
// Q3 -> R3 -> S3 -> T3 -> U3 -> G1
//rule(w(a(3,Q)), g0, set(a(3,R)));     // 5
rule(w(a(3,Q)), g0, primitive(SetWithCost3, w(a(3,R))));
rule(w(a(3,R)), g0, set(a(3,S)));     // 6
rule(w(a(3,S)), g0, set(a(3,T)));     // 7
rule(w(a(3,T)), g0, set(a(3,U)));     // 8
rule(w(a(3,U)), g0, set(a(1,G)));     // 9

setRulesName("g1");
// P1 -> Q1 -> R1 -> S1
rule(w(a(1,P)), g1, set(a(1,Q)));     // 1
//rule(w(a(1,Q)), g1, set(a(1,R)));     // 2
rule(w(a(1,Q)), g1, primitive(SetWithCost1, w(a(1,R))));
rule(w(a(1,R)), g1, set(a(1,S)));     // 3
// P1 -> call(R2) -> S2
rule(w(a(1,P)), g1, call(g2));        // 4
rule(w(a(2,R)), g1, set(a(2,S)));     // 5
// exit(Q3)
rule(w(a(1,P)), g1, exit(a(3,Q)));    // 6

setRulesName("g2");
//rule(w(a(1,P)), g2, set(a(2,R)));     // 1
rule(w(a(1,P)), g2, primitive(SetWithCost2, w(a(2,R))));
```

モンテカルロで学習 beta=1
P→R2 のコスト-0.5

alpha=0.001
beta=1
約10万エピソード



エピソードを最も低コストで終了できるルートが
高確率で選択される

Alice
returnRule.q: 0.0
returnRule.base: 0.0
startRule.q: 0.0
startRule.base: -0.9999702
r.base:

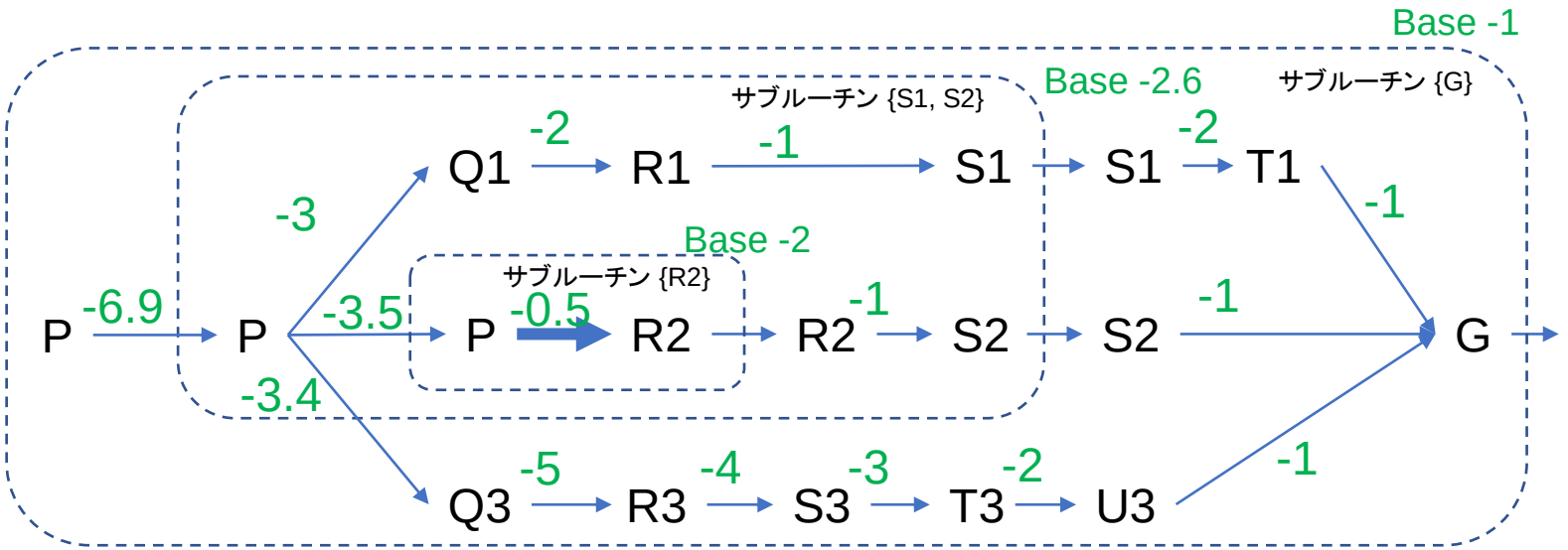
0	g0.1	-3.3839872
1	g0.2	0.0
2	g0.3	0.0
3	g0.4	0.0
4	g0.5	0.0
5	g0.6	0.0
6	g0.7	0.0
7	g0.8	0.0
8	g0.9	0.0
9	g1.1	0.0
10	g1.2	0.0
11	g1.3	0.0
12	g1.4	-4.9997616
13	g1.5	0.0
14	g1.6	0.0
15	g2.1	0.0

r.q:

0	g0.1	-6.781405
1	g0.2	-1.9999702
2	g0.3	-1.0
3	g0.4	-1.0
4	g0.5	-4.999791
5	g0.6	-3.9999104
6	g0.7	-2.9999108
7	g0.8	-1.9999702
8	g0.9	-1.0
9	g1.1	-3.623542
10	g1.2	-2.623542
11	g1.3	-1.6235423
12	g1.4	-3.125698
13	g1.5	-0.6256995
14	g1.6	-3.6239288
15	g2.1	-0.5002682

Sarsa で学習 beta=1
P→R2 のコスト-0.5

alpha=0.001
beta=1
約10万エピソード



エピソードを最も低コストで終了できるルートでないルート P→Q1 が
高確率で選択される

Alice
returnRule.q: 0.0
returnRule.base: 0.0
startRule.q: 0.0
startRule.base: -0.9999702
r.base:

0	g0.1	-2.6175258
1	g0.2	0.0
2	g0.3	0.0
3	g0.4	0.0
4	g0.5	0.0
5	g0.6	0.0
6	g0.7	0.0
7	g0.8	0.0
8	g0.9	0.0
9	g1.1	0.0
10	g1.2	0.0
11	g1.3	0.0
12	g1.4	-1.9999106
13	g1.5	0.0
14	g1.6	0.0
15	g2.1	0.0

r.q:

0	g0.1	-6.858474
1	g0.2	-1.9999106
2	g0.3	-0.9999702
3	g0.4	-0.9999702
4	g0.5	-4.9994345
5	g0.6	-3.9996724
6	g0.7	-2.9997916
7	g0.8	-1.9999106
8	g0.9	-0.9999702
9	g1.1	-2.9997916
10	g1.2	-1.9999106
11	g1.3	-0.9999702
12	g1.4	-3.4997764
13	g1.5	-0.9999702
14	g1.6	-3.3881714
15	g2.1	-0.4999851

現時点での問題点

- Sarsa 版ではサブルーチン終了後の報酬が行動価値に反映されない
- モンテカルロ版はマルチタスクへの対応が不完全

- 収束に至るまでの価値の不安定性
- パラメタ共有がうまくいかないサブルーチンの扱い
- 生物学的妥当性の検討が不十分

まとめと今後

- 本発表
  - RGoal を複数の出口を持てるように拡張
  - 例外終了の機能を追加
  - ２つの学習アルゴリズムの実装： Sarsa 、モンテカルロ法（改良の余地あり）
  - １０個ほどのテスト用タスクで動作確認（まだ不十分）
- 今後
  - マルチタスクでのテスト
  - 性能の定量的評価
  - 他の階層型強化学習との詳細な比較
  - 複雑なデモタスクで提案アルゴリズムの有用性を示す
    - エージェントが行動計画を立ててから行動し、
      結果によって行動計画戦略を改善するようなタスクを計画中

以上

旧 RGoal [Ichisugi+ ICANN 2017]

- RGoal は深さの制限なくサブルーチン呼び出しができる階層型強化学習
- エージェントは内部状態としてサブゴール g とサブゴールのスタックを持つ
- 任意の状態からサブゴール g に向かう方策を「サブルーチン g 」と呼ぶ
- エージェントは普通の行動の他に、サブルーチン呼び出し call(g) を常に選択可能
- 行動価値関数（ゴールに到達するまでの報酬の総和）をサブゴール到達の前と後に分割し、サブゴール到達前の報酬の総和 Q(s,g,a) だけを学習
- RGoal はマルチタスク環境で近似解を高速に得ることと、
  神経科学的妥当性の高さが設計の目標

- 旧 RGoal では g は単一の状態であったが、本研究では g を状態の集合とする

Pro5Lang で行われる様々なパラメタ共有 (Sarsa, モンテカルロ法 共通)

- サブタスク共有
- ゴールの状態の抽象化
  - {g1,g2,g3,...} = g
  - V(g1),V(g2),... ≒ V(g) ≡ 0
- 行動ルールの抽象化
  - { (s1,g1,a1), (s1,g2,a1), ... } = rule
  - 行動価値関数のパラメタ共有
    - Q(s1,g1,a1), Q(s2,g2,a2), ... ≒ Q(rule)
  - 呼び出し文脈ごとの基準値の学習パラメタの共有
    - Base(s1,g1,a1), Base(s2,g2,a2), ... ≒ Base(rule)

Pro5Lang のテーブル圧縮（価値関数の近似）

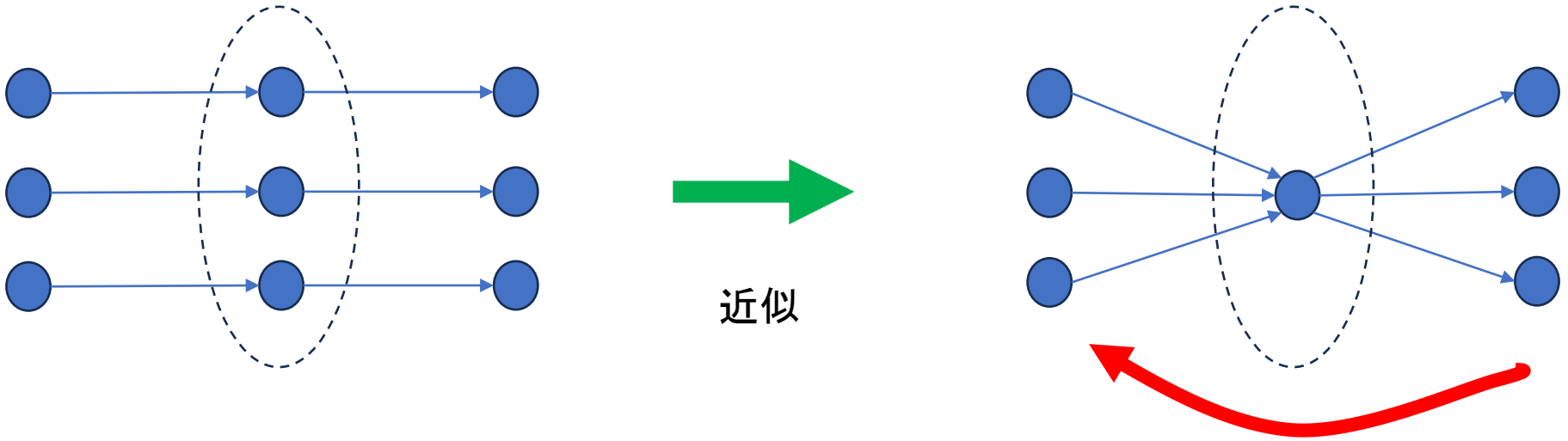
rule(P(1), g, set(Q(1)))

rule(P(2), g, set(Q(2)))

rule(P(3), g, set(Q(3)))

抽象化
→

rule(P(x), g, set(Q(x)))



Sarsa(0) では価値が伝わらない
Sarsa(λ), モンテカルロ法なら伝わる

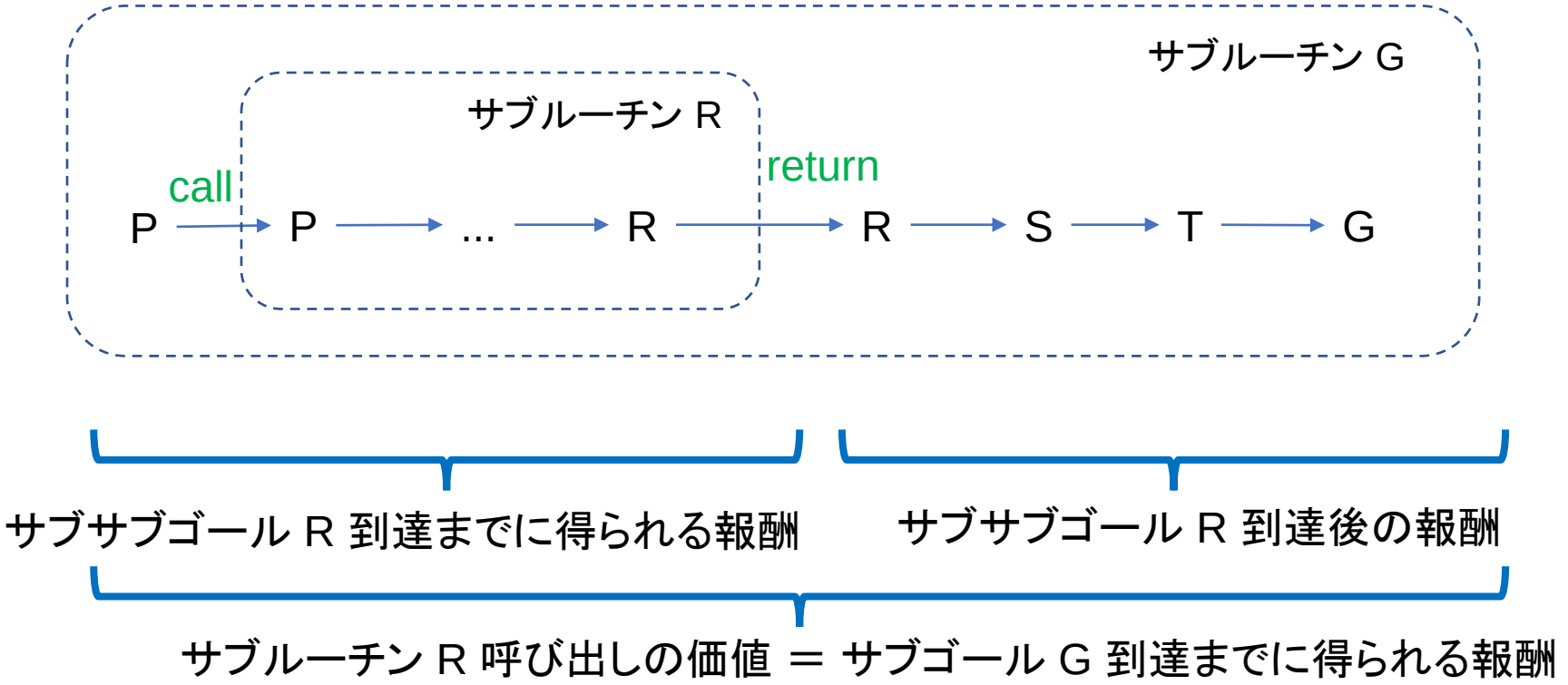
プログラム獲得のためのさまざまな機構

- 自動応答機構：
  - 刺激に対して自動的に応答（目立つ物体に視線を向ける、など）
  - すべての行動の最初のきっかけとなる。（？）

今回の発表

- 行動価値評価機構：
  - 価値に基づいて行動し、行動結果の価値を評価し、行動価値を更新
- 帰納推論機構：
  - 経験履歴に基づいて、汎用性の高い行動ルールを獲得
- 演繹推論機構：
  - 汎用的な行動ルールや宣言的知識を組み合わせる
  - 模倣や対話を通じた知識獲得

旧 RGoal の価値関数分解とその問題点



Cf. MAX 価値関数分解
[Dietterich 2000]

サブルーチンを抜けるための状態（サブゴール）は1つだけという制限があった

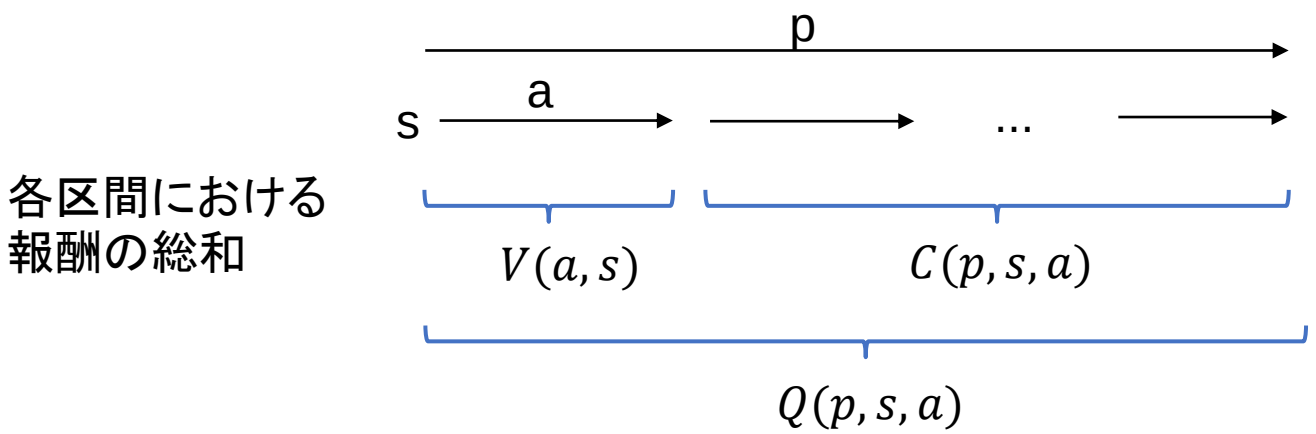
MAXQ 価値関数分解[Dietterich 2000]

$$Q(p,s,a) = V(a,s) + C(p,s,a)$$

p は親タスク、a は子タスクまたはプリミティブな行動。
状態（サブゴール）ではなく行動（サブルーチン）の価値に着目した式。
V(a,s) は親タスク p に依存しないので**タスク間で共有できる**。
ただし V(a,s) は以下の式で再帰的に計算する必要がある。

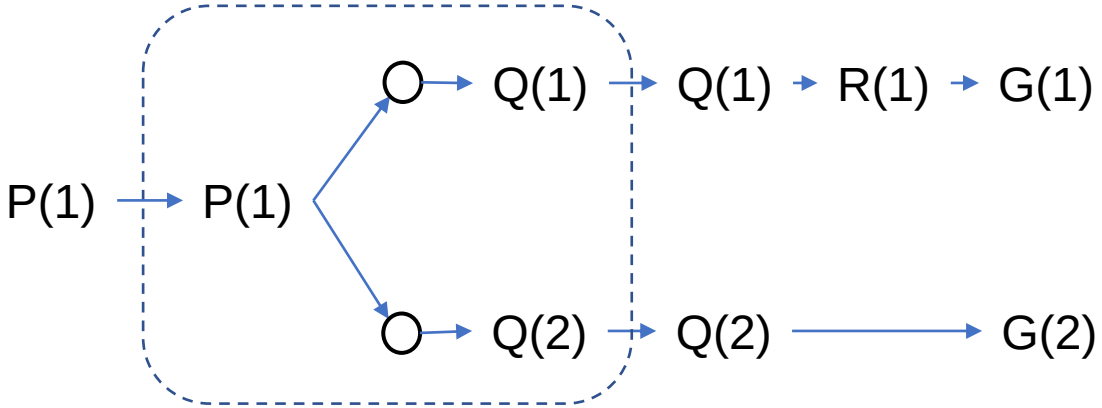
$$V(p,s) = \begin{cases} \max_a Q(p,s,a) & if\ p\ is\ composite \\ V(p,s) & if\ p\ is\ primitive \end{cases}$$

$$Q(p,s,a) = V(a,s) + C(p,s,a)$$



テスト用タスクと
実行結果

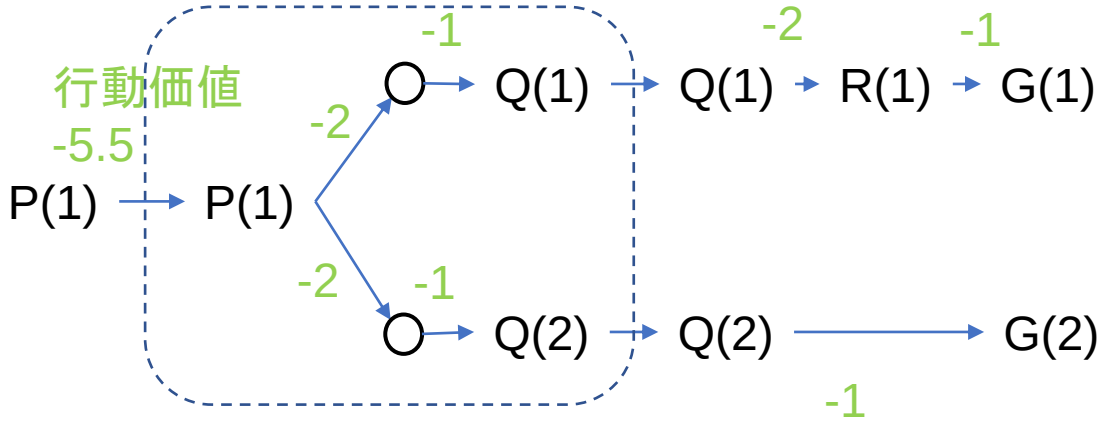
テスト１



Pro5Lang ソースコード

```
setRulesName("g0");
rule(w(a(1,P)), g0, call(g1));    // 1
// route 1
rule(w(a(1,Q)), g0, set(a(1,R))); // 2
rule(w(a(1,R)), g0, set(a(1,G))); // 3
// route 2
rule(w(a(2,Q)), g0, set(a(2,G))); // 4

setRulesName("g1");
rule(w(a(1,P)), g1, findItemA(Stone));  // 1
rule(w(a(Now,Room1,Alice,IsAwareOf,Stone,0)), g1, set(a(1,Q)));   // 1
rule(w(a(Now,Room1,Alice,IsAwareOfNo,Stone,0)), g1, set(a(2,Q))); // 2
```

テスト１　学習結果 Sarsa




```
setRulesName("g0");
rule(w(a(1,P)), g0, call(g1));    // 1
// route 1
rule(w(a(1,Q)), g0, set(a(1,R))); // 2
rule(w(a(1,R)), g0, set(a(1,G))); // 3
// route 2
rule(w(a(2,Q)), g0, set(a(2,G))); // 4

setRulesName("g1");
rule(w(a(1,P)), g1, findItemA(Stone));  // 1
rule(w(a(Now,Room1,Alice,IsAwareOf,Stone,0)), g1, set(a(1,Q)));   // 1
rule(w(a(Now,Room1,Alice,IsAwareOfNo,Stone,0)), g1, set(a(2,Q))); // 2
```

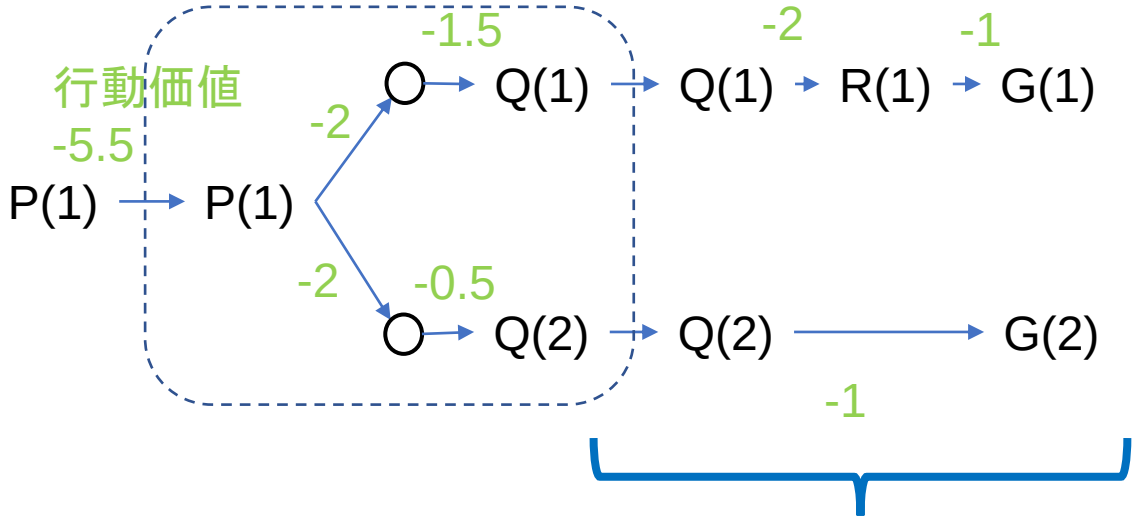
Q(P(1), G(+), call(Q(+)))
= -1 + -2 + -1 + (1/2 * (-2) + 1/2 * (-1))
= -5.5

alpha=0.001
beta=1e-5
約10万エピソード学習

0	g0.1	-5.481687
1	g0.2	-1.9999106
2	g0.3	-0.9999702
3	g0.4	-0.9999702
4	g1.1	-1.9999106
5	g1.2	-0.9999702
6	g1.3	-0.9999702

beta が小さいので
行動価値によらず
ほぼ等確率で行動選択

テスト１　学習結果　モンテカルロ




```
setRulesName("g0");
rule(w(a(1,P)), g0, call(g1));    // 1
// route 1
rule(w(a(1,Q)), g0, set(a(1,R))); // 2
rule(w(a(1,R)), g0, set(a(1,G))); // 3
// route 2
rule(w(a(2,Q)), g0, set(a(2,G))); // 4

setRulesName("g1");
rule(w(a(1,P)), g1, findItemA(Stone));  // 1
rule(w(a(Now,Room1,Alice,IsAwareOf,Stone,0)), g1, set(a(1,Q)));   // 2
rule(w(a(Now,Room1,Alice,IsAwareOfNo,Stone,0)), g1, set(a(2,Q))); // 3
```

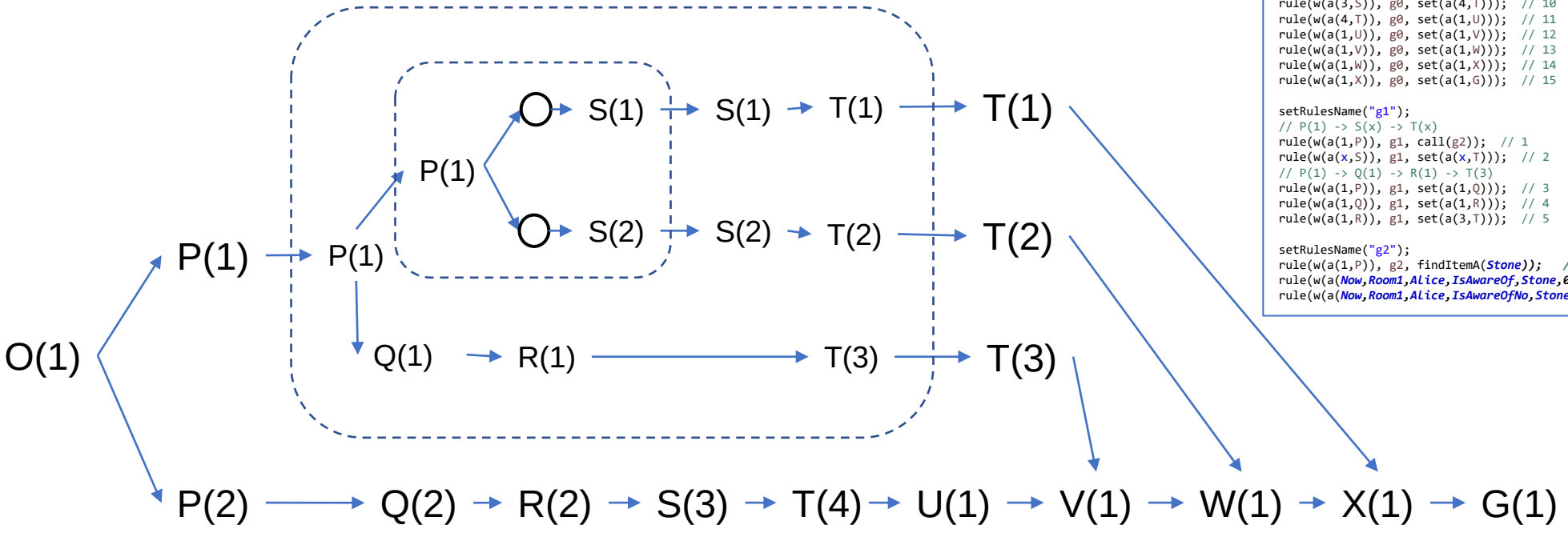
基準値(サブルーチンを正常終了してから
エピソード終了までの累積報酬期待値)
(-3 + -2) / 2 = -2.5

alpha=0.001
beta=1e-5
約10万エピソード学習

0	g0.1	-5.4946895
1	g0.2	-1.9999702
2	g0.3	-1.0
3	g0.4	-1.0
4	g1.1	-2.01283
5	g1.2	-1.5154262
6	g1.3	-0.5170868

Q(1) で終了する行動の相対価値：　-4 - (-2.5) = -1.5
Q(2) で終了する行動の相対価値：　-3 - (-2.5) = -0.5

テスト２

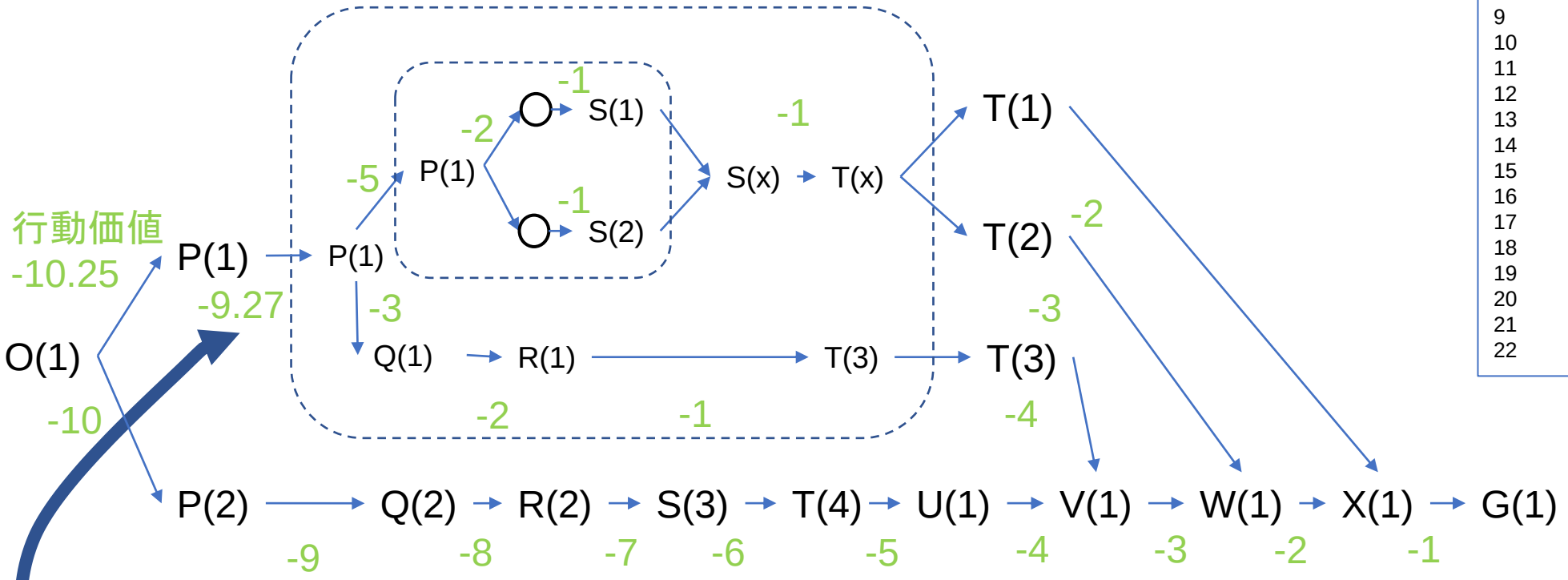



```
setRulesName("g0");
rule(w(a(1,O)), g0, set(a(1,P)));  // 1
rule(w(a(1,O)), g0, set(a(2,P)));  // 2
// {} -> P(1) -> call(T(+))
rule(w(a(1,P)), g0, call(g1));  // 3
// {} -> P(1) -> T(1) -> X(1)
rule(w(a(1,T)), g0, set(a(1,X)));  // 4
// {} -> P(1) -> T(2) -> W(1)
rule(w(a(2,T)), g0, set(a(1,W)));  // 5
// {} -> P(1) -> T(3) -> V(1)
rule(w(a(3,T)), g0, set(a(1,V)));  // 6
// {} -> P(2) -> Q(1) -> R(1) -> ... -> X(1) -> G(1)
rule(w(a(2,P)), g0, set(a(2,Q)));  // 7
rule(w(a(2,Q)), g0, set(a(2,R)));  // 8
rule(w(a(2,R)), g0, set(a(3,S)));  // 9
rule(w(a(3,S)), g0, set(a(4,T)));  // 10
rule(w(a(4,T)), g0, set(a(1,U)));  // 11
rule(w(a(1,U)), g0, set(a(1,V)));  // 12
rule(w(a(1,V)), g0, set(a(1,W)));  // 13
rule(w(a(1,W)), g0, set(a(1,X)));  // 14
rule(w(a(1,X)), g0, set(a(1,G)));  // 15

setRulesName("g1");
// P(1) -> S(x) -> T(x)
rule(w(a(1,P)), g1, call(g2));  // 1
rule(w(a(x,S)), g1, set(a(x,T)));  // 2
// P(1) -> Q(1) -> R(1) -> T(3)
rule(w(a(1,P)), g1, set(a(1,Q)));  // 3
rule(w(a(1,Q)), g1, set(a(1,R)));  // 4
rule(w(a(1,R)), g1, set(a(3,T)));  // 5

setRulesName("g2");
rule(w(a(1,P)), g2, findItemA(Stone));   // 1
rule(w(a(Now,Room1,Alice,IsAwareOf,Stone,0)), g2, set(a(1,S)));   // 2
rule(w(a(Now,Room1,Alice,IsAwareOfNo,Stone,0)), g2, set(a(2,S)));  // 3
```

テスト２の学習結果１　Sarsa

alpha=0.001
beta=1e-5
約10万エピソード学習

0	g0.1	-10.250875
1	g0.2	-9.9977665
2	g0.3	-9.271175
3	g0.4	-1.9999106
4	g0.5	-2.9997916
5	g0.6	-3.9996724
6	g0.7	-8.998243
7	g0.8	-7.998719
8	g0.9	-6.9989576
9	g0.10	-5.999196
10	g0.11	-4.9994345
11	g0.12	-3.9996724
12	g0.13	-2.9997916
13	g0.14	-1.9999106
14	g0.15	-0.9999702
15	g1.1	-4.9995832
16	g1.2	-0.9999702
17	g1.3	-2.9997916
18	g1.4	-1.9999106
19	g1.5	-0.9999702
20	g2.1	-1.9999106
21	g2.2	-0.9999702
22	g2.3	-0.9999702

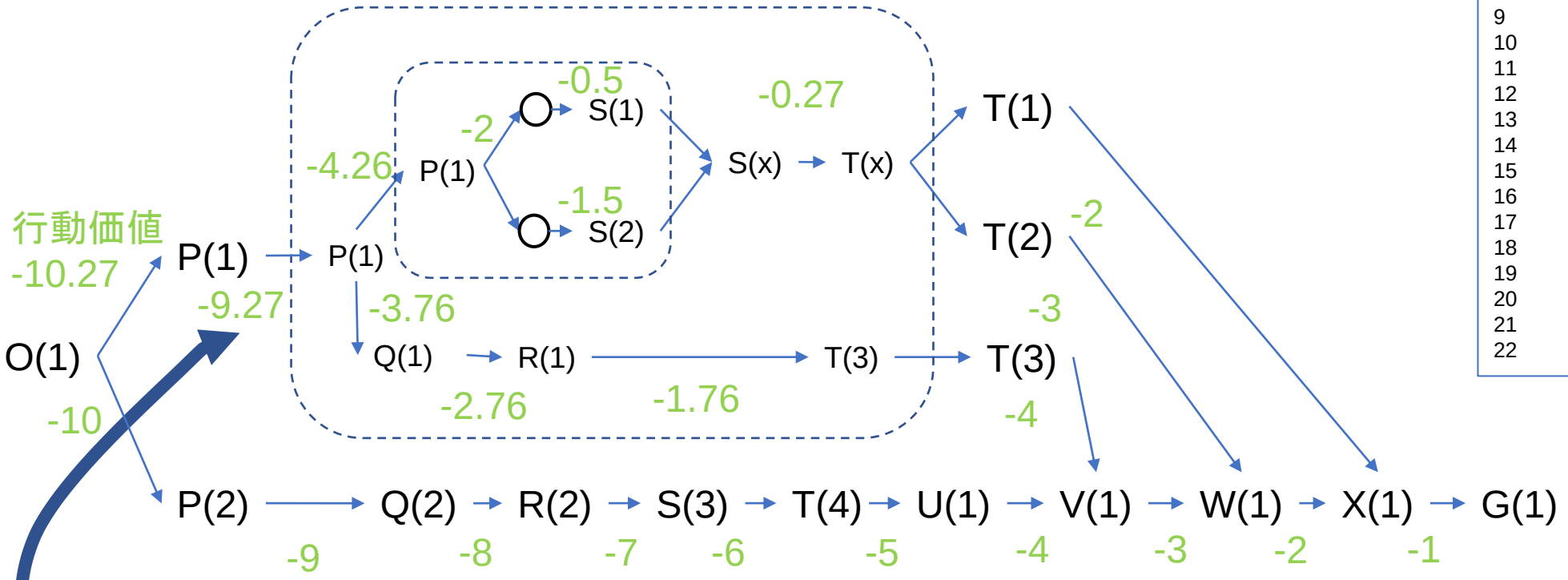


Q(P(1), G(+), call(T(+))
= -1 + ((0.5 * -5) + (0.5 * -3)) + -1 + ((0.25 * -2) + (0.25 * -3) + (0.5 * -4))
= -9.25

テスト２の学習結果１　モンテカルロ

alpha=0.001
beta=1e-5
約10万エピソード学習

0	g0.1	-10.268073
1	g0.2	-9.999554
2	g0.3	-9.268073
3	g0.4	-1.9999702
4	g0.5	-2.9999108
5	g0.6	-3.9999104
6	g0.7	-8.999554
7	g0.8	-7.999791
8	g0.9	-6.999791
9	g0.10	-5.999791
10	g0.11	-4.999791
11	g0.12	-3.9999104
12	g0.13	-2.9999108
13	g0.14	-1.9999702
14	g0.15	-1.0
15	g1.1	-4.2689114
16	g1.2	-0.26891258
17	g1.3	-3.757307
18	g1.4	-2.757307
19	g1.5	-1.7573085
20	g2.1	-2.009013
21	g2.2	-0.49826962
22	g2.3	-1.4974338

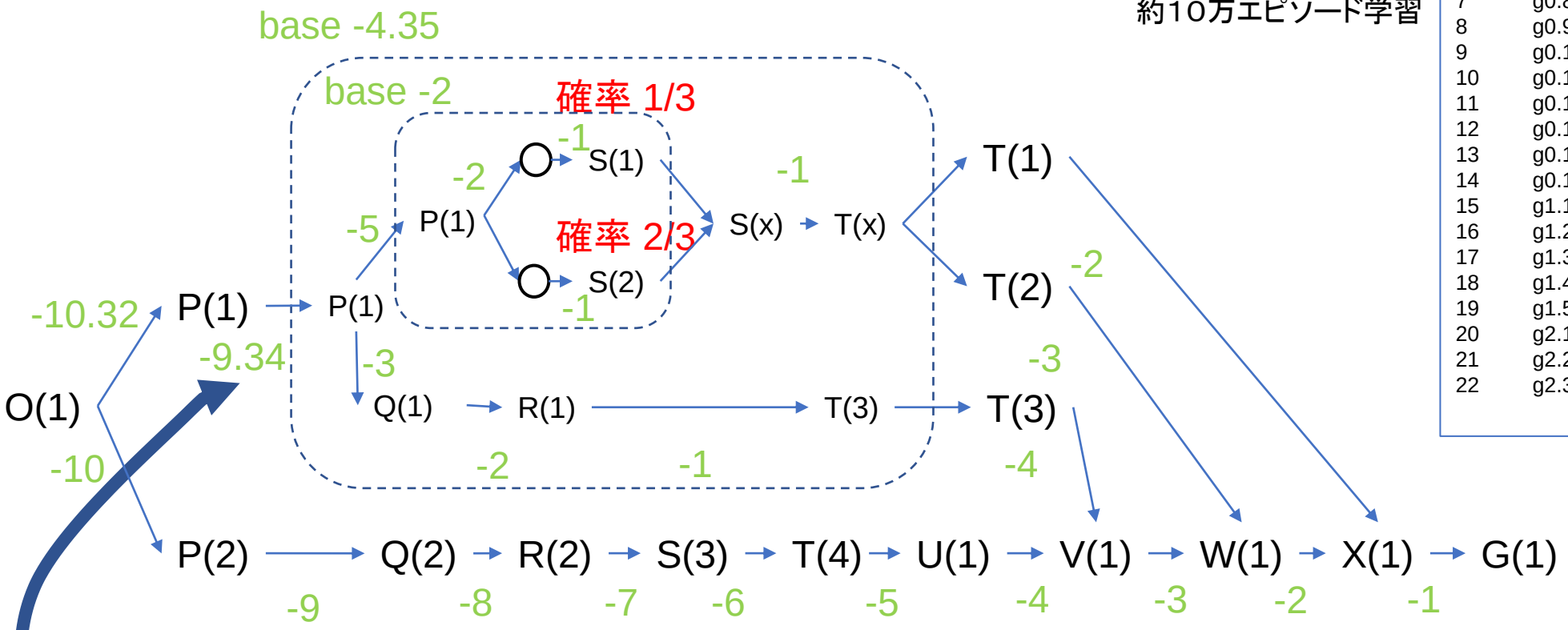


Q(P(1), G(+), call(T(+))
= -1 + ((0.5 * -5) + (0.5 * -3)) + -1 + ((0.25 * -2) + (0.25 * -3) + (0.5 * -4))
= -9.25

テスト２の学習結果２　Sarsa

alpha=0.001
beta=1e-5
約10万エピソード学習

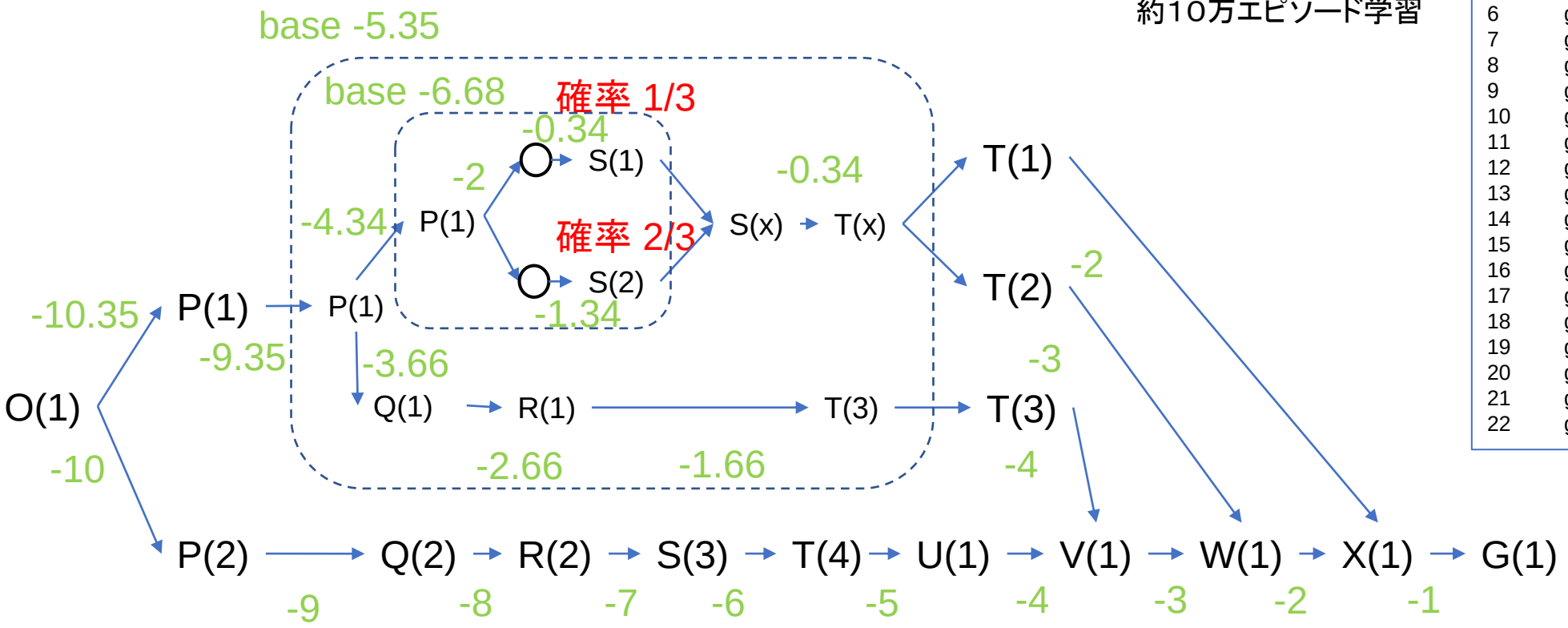
0	g0.1	-10.324317
1	g0.2	-9.9977665
2	g0.3	-9.346801
3	g0.4	-1.9994962
4	g0.5	-2.9997916
5	g0.6	-3.9996724
6	g0.7	-8.998243
7	g0.8	-7.998719
8	g0.9	-6.9989576
9	g0.10	-5.999196
10	g0.11	-4.9994345
11	g0.12	-3.9996724
12	g0.13	-2.9997916
13	g0.14	-1.9999106
14	g0.15	-0.9999702
15	g1.1	-4.999531
16	g1.2	-0.9999702
17	g1.3	-2.9997916
18	g1.4	-1.9999106
19	g1.5	-0.9999702
20	g2.1	-1.9998579
21	g2.2	-0.9997723
22	g2.3	-0.9999702



Q(P(1), G(+), call(T(+))
= -1 + ((0.5 * -5) + (0.5 * -3)) + -1 + ((0.5 * (1/3) * -2) + (0.5 * (2/3) * -3) + (0.5 * -4))
= -9.333

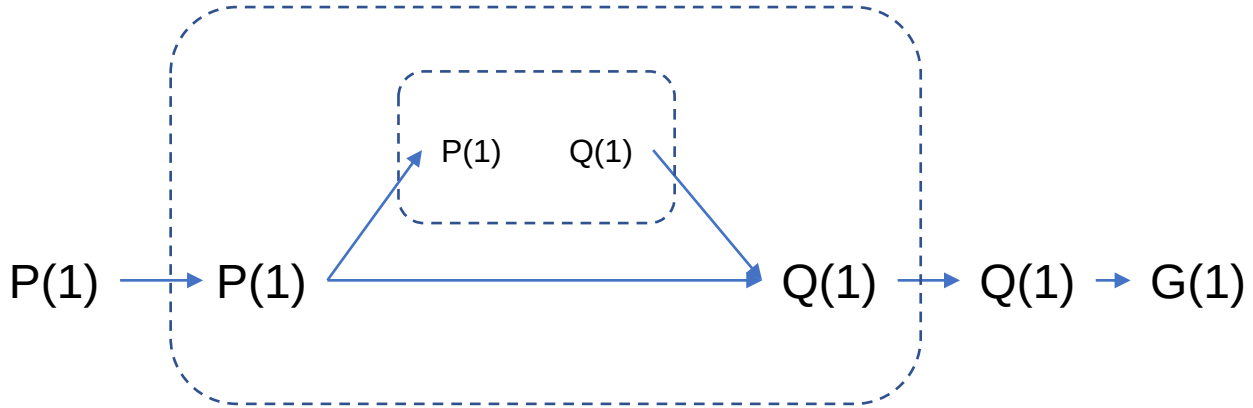
テスト２の学習結果２　モンテカルロ法

alpha=0.001
beta=1e-5
約10万エピソード学習



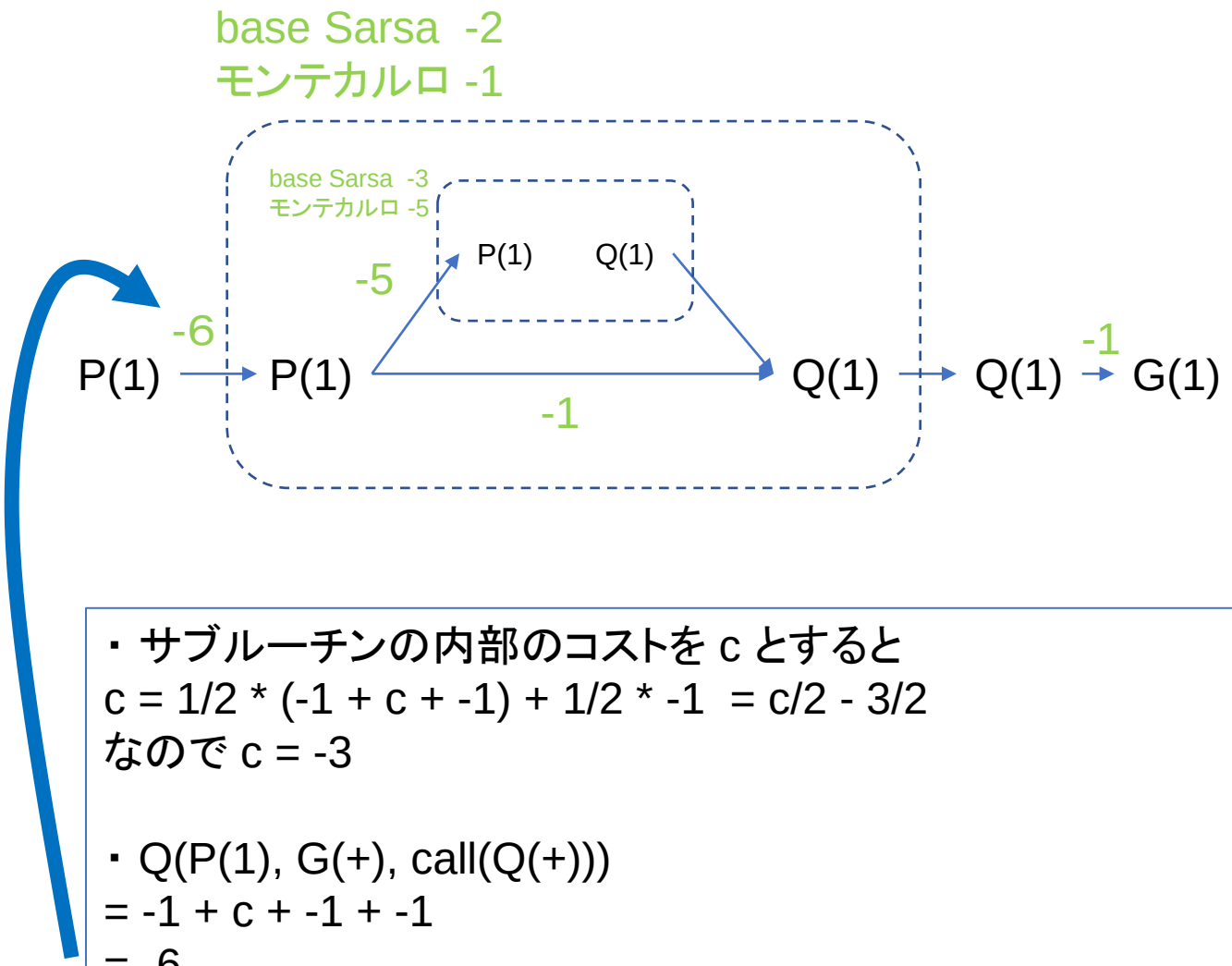
0	g0.1	-10.34963
1	g0.2	-9.999554
2	g0.3	-9.34963
3	g0.4	-1.9995744
4	g0.5	-2.9999108
5	g0.6	-3.9999104
6	g0.7	-8.999554
7	g0.8	-7.999791
8	g0.9	-6.999791
9	g0.10	-5.999791
10	g0.11	-4.999791
11	g0.12	-3.9999104
12	g0.13	-2.9999108
13	g0.14	-1.9999702
14	g0.15	-1.0
15	g1.1	-4.3444223
16	g1.2	-0.3444168
17	g1.3	-3.6608129
18	g1.4	-2.6608129
19	g1.5	-1.6608157
20	g2.1	-2.0209603
21	g2.2	-0.33913013
22	g2.3	-1.3389684

テスト３　再帰呼び出し




```
setRulesName("g0");
rule(w(a(1,P)), g0, call(g1));  // 1
rule(g1, g0, set(a(1,G)));      // 2

setRulesName("g1");
rule(w(a(1,P)), g1, call(g1));    // 1
rule(w(a(1,P)), g1, set(a(1,Q))); // 2
```

テスト３　再帰呼び出し　学習結果




```
setRulesName("g0");
rule(w(a(1,P)), g0, call(g1));  // 1
rule(g1, g0, set(a(1,G)));      // 2

setRulesName("g1");
rule(w(a(1,P)), g1, call(g1));    // 1
rule(w(a(1,P)), g1, set(a(1,Q))); // 2
```

・サブルーチンの内部のコストを c とすると
c = 1/2 * (-1 + c + -1) + 1/2 * -1 = c/2 - 3/2
なので c = -3

・Q(P(1), G(+), call(Q(+)))
= -1 + c + -1 + -1
= -6

alpha=0.001
beta=1e-5
約10万エピソード学習

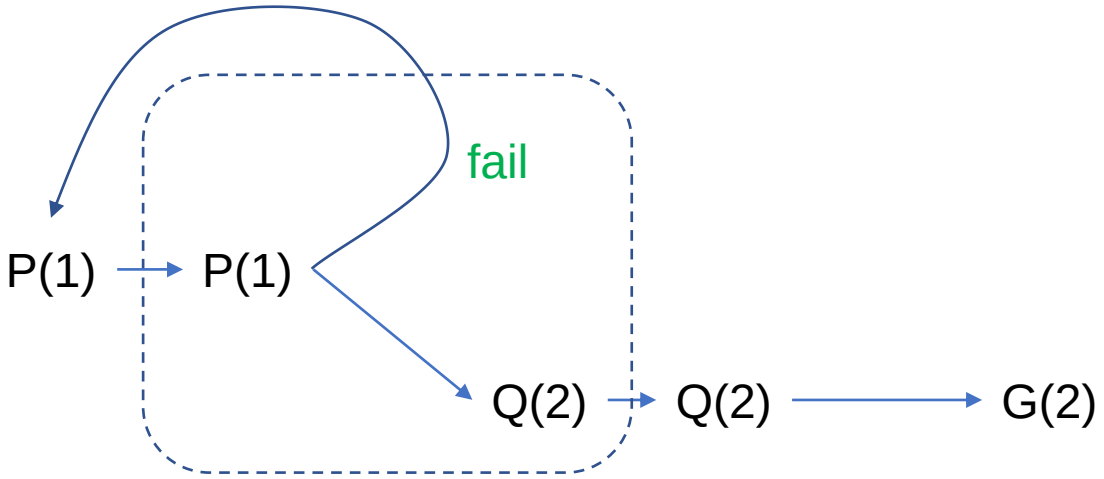
Sarsa

0	g0.1	-6.059001
1	g0.2	-0.9999702
2	g1.1	-4.9841566
3	g1.2	-0.9999702

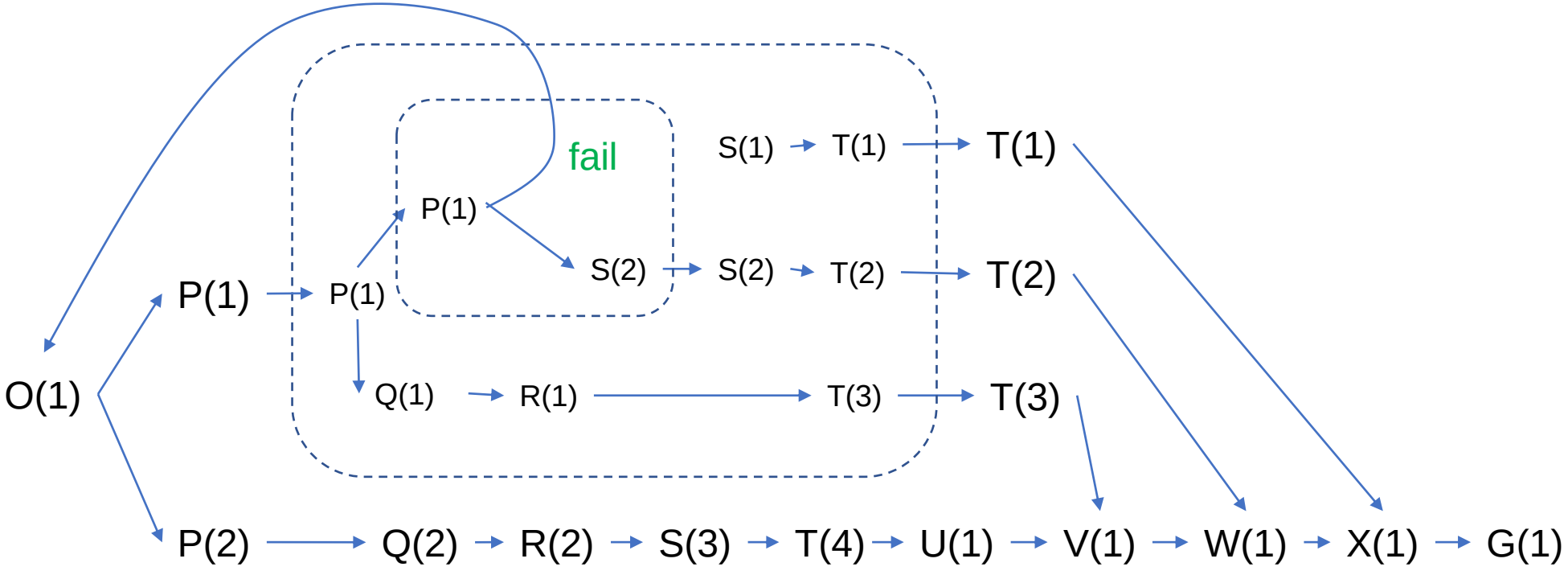
モンテカルロ

0	g0.1	-6.0034614
1	g0.2	-1.0
2	g1.1	-5.206631
3	g1.2	-0.95455545

テスト４ fail（未実装）

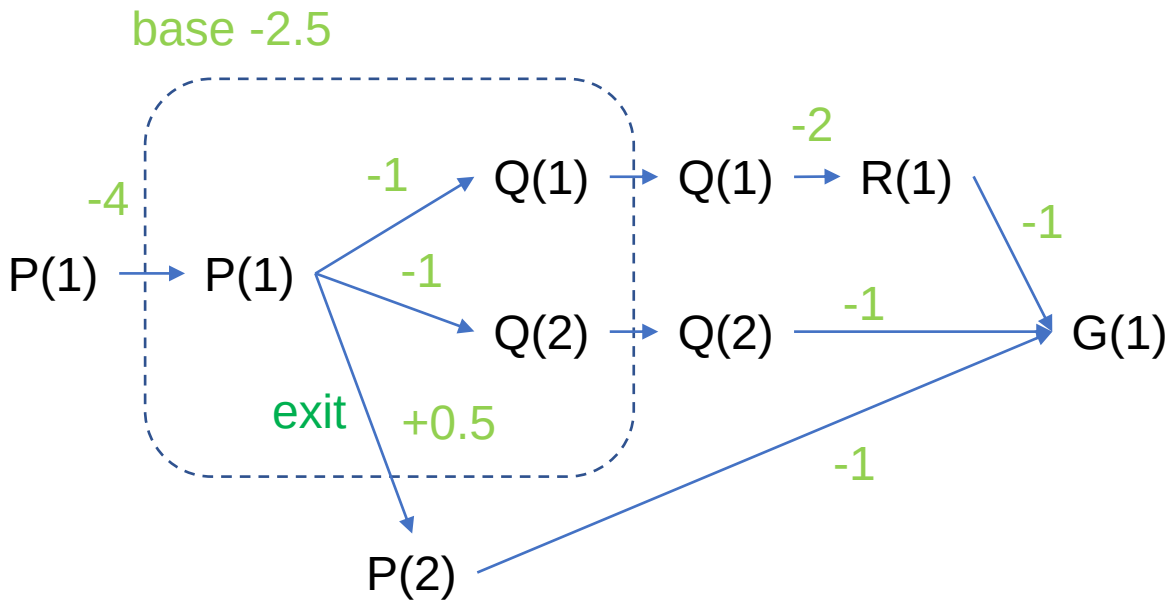


テスト５ fail（未実装）



テスト６ exit 学習結果 Sarsa

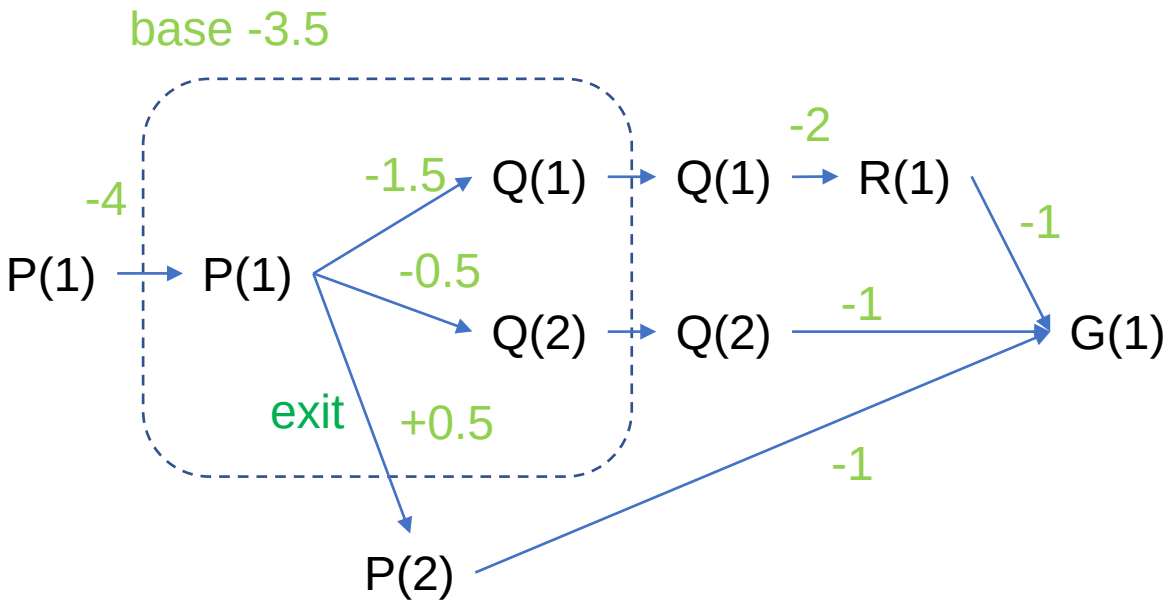
```
setRulesName("g0");
rule(w(a(1,P)), g0, call(g1));    // 1
// route 1
rule(w(a(1,Q)), g0, set(a(1,R))); // 2
rule(w(a(1,R)), g0, set(a(1,G))); // 3
// route 2
rule(w(a(2,Q)), g0, set(a(1,G))); // 4
// route 3
rule(w(a(2,P)), g0, set(a(1,G))); // 5

setRulesName("g1");
rule(w(a(1,P)), g1, set(a(1,Q)));  // 1
rule(w(a(1,P)), g1, set(a(2,Q)));  // 2
rule(w(a(1,P)), g1, exit(a(2,P)));  // 3
```



alpha=0.001
beta=1e-5
約10万エピソード学習

0	g0.1	-4.005056
1	g0.2	-1.9999106
2	g0.3	-0.9999702
3	g0.4	-0.9999702
4	g0.5	-0.9999702
5	g1.1	-0.9999702
6	g1.2	-0.9999702
7	g1.3	0.5003009

テスト６ exit 学習結果
モンテカルロ

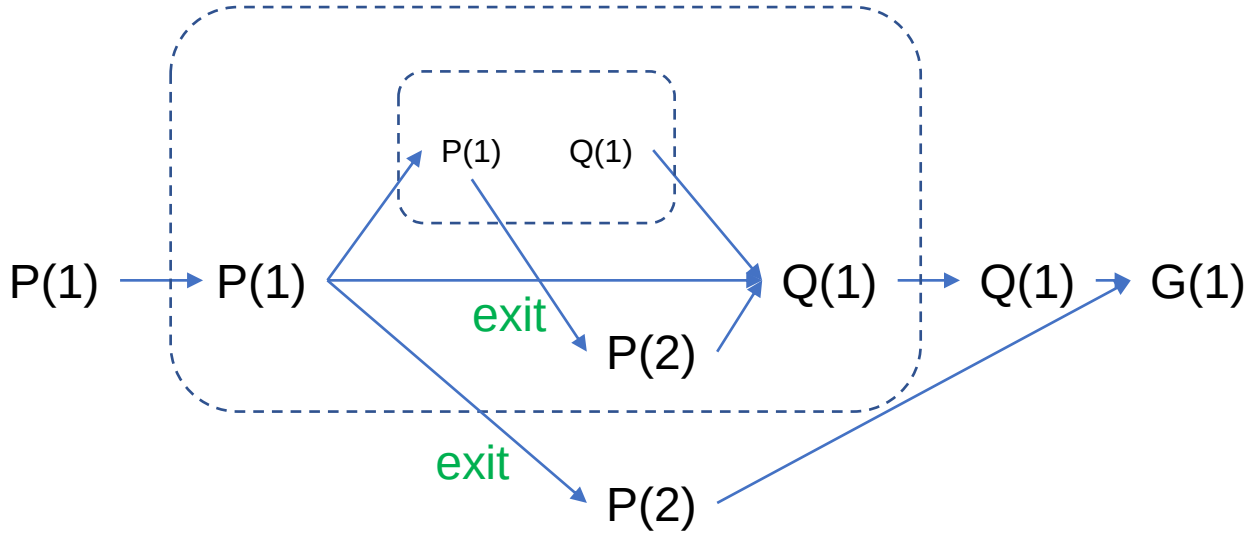



```
setRulesName("g0");
rule(w(a(1,P)), g0, call(g1));    // 1
// route 1
rule(w(a(1,Q)), g0, set(a(1,R))); // 2
rule(w(a(1,R)), g0, set(a(1,G))); // 3
// route 2
rule(w(a(2,Q)), g0, set(a(1,G))); // 4
// route 3
rule(w(a(2,P)), g0, set(a(1,G))); // 5

setRulesName("g1");
rule(w(a(1,P)), g1, set(a(1,Q)));  // 1
rule(w(a(1,P)), g1, set(a(2,Q)));  // 2
rule(w(a(1,P)), g1, exit(a(2,P)));  // 3
```

alpha=0.001
beta=1e-5
約10万エピソード学習

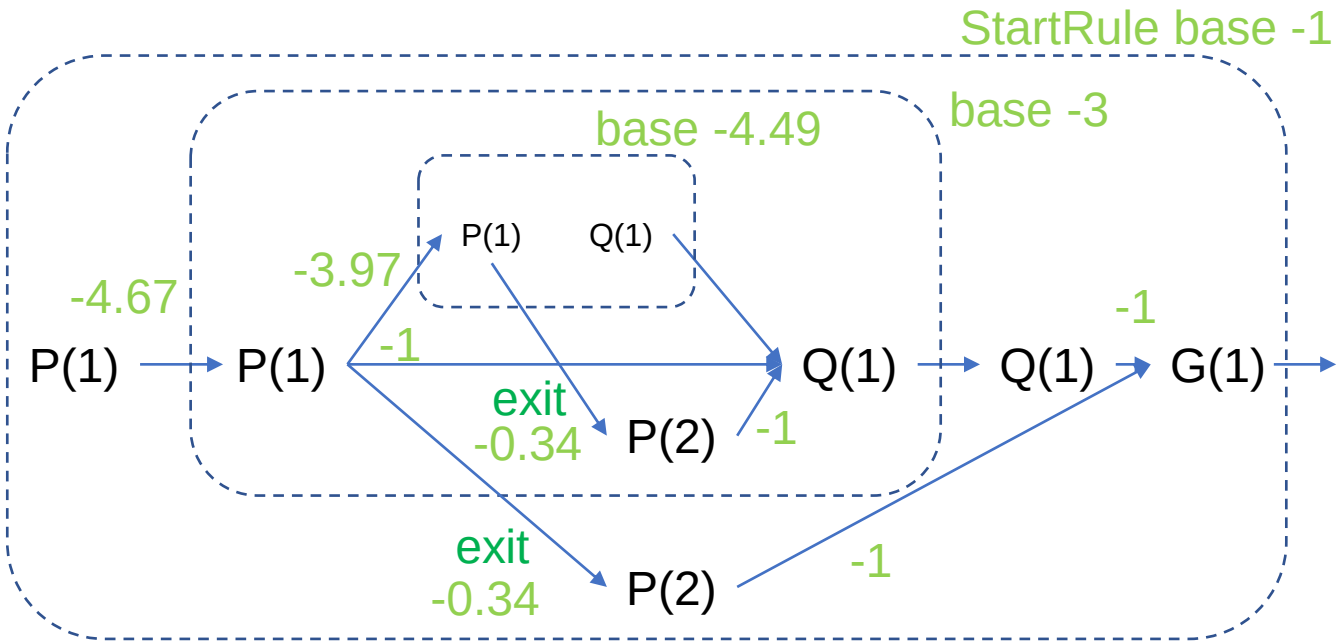
0	g0.1	-3.9966471
1	g0.2	-1.9999702
2	g0.3	-1.0
3	g0.4	-1.0
4	g0.5	-1.0
5	g1.1	-1.4994103
6	g1.2	-0.5003828
7	g1.3	0.5003285

テスト７ exit 再帰呼び出し




```
setRulesName("g0");
rule(w(a(1,P)), g0, call(g1));  // 1
rule(g1, g0, set(a(1,G)));      // 2
rule(w(a(2,P)), g0, set(a(1,G))); // 3

setRulesName("g1");
rule(w(a(1,P)), g1, call(g1));     // 1
rule(w(a(1,P)), g1, set(a(1,Q)));  // 2
rule(w(a(1,P)), g1, exit(a(2,P))); // 3
rule(w(a(2,P)), g1, set(a(1,Q))); // 4
```

テスト７ exit 再帰呼び出し 学習結果
モンテカルロ法（Sarsa も同じ結果）



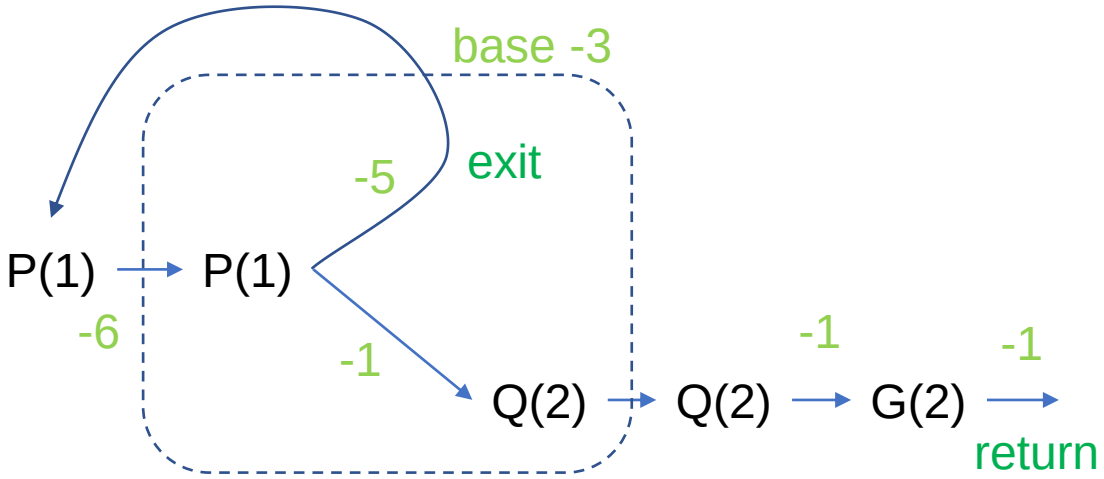

```
setRulesName("g0");
rule(w(a(1,P)), g0, call(g1));  // 1
rule(g1, g0, set(a(1,G)));      // 2
rule(w(a(2,P)), g0, set(a(1,G))); // 3

setRulesName("g1");
rule(w(a(1,P)), g1, call(g1));     // 1
rule(w(a(1,P)), g1, set(a(1,Q)));  // 2
rule(w(a(1,P)), g1, exit(a(2,P))); // 3
rule(w(a(2,P)), g1, set(a(1,Q))); // 4
```

alpha = 0.001
beta = 1e-4
10万エピソード

0	g0.1	-4.6737857
1	g0.2	-0.9999732
2	g0.3	-0.9999732
3	g1.1	-3.9732273
4	g1.2	-0.99507886
5	g1.3	-0.34169364
6	g1.4	-0.98401076

・内側の call(g1) の価値 -3.97 が正しいとする
　内側の call(g1) の価値: -1 + (-3.97-1)/3 + (-1-1)/3 + (-1-1)/3 = -3.99
　外側の call(g1) の価値: -1 + (-3.97-2)/3 + (-1-2)/3 + (-1-1)/3 = -4.65
となり**ほぼ整合性は取れている**
・exit の価値については未検証

テスト８ exit してやり直し
モンテカルロ法（Sarsa も同じ結果）

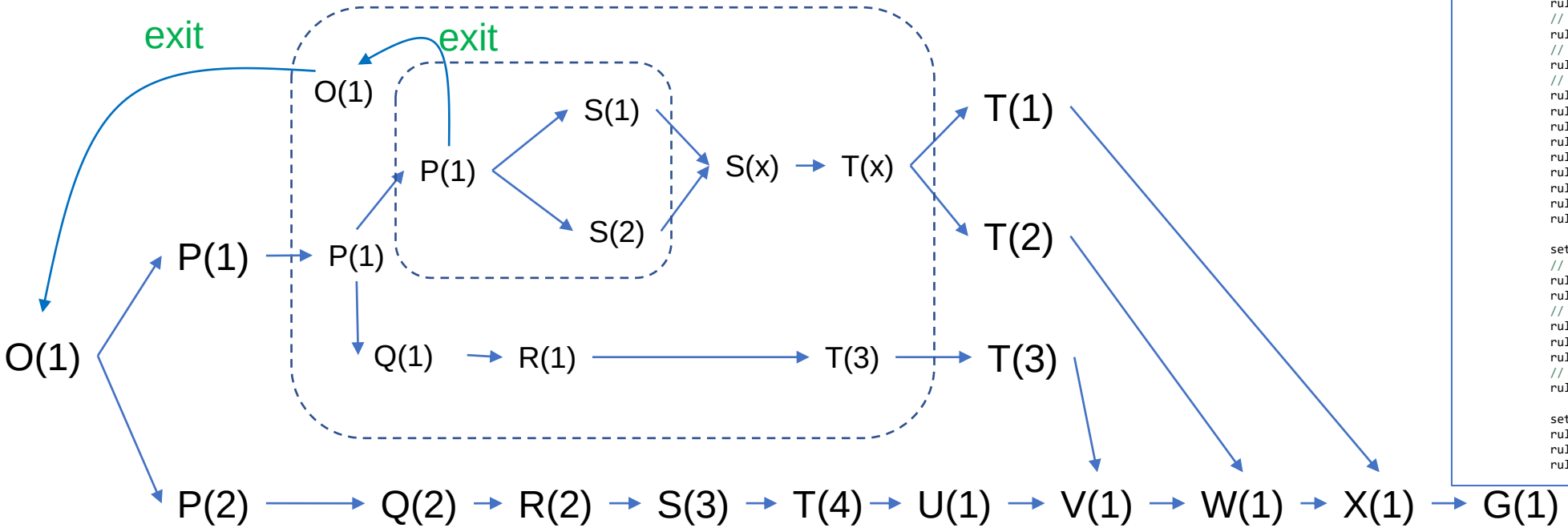



```
setRulesName("g0");
rule(w(a(1,P)), g0, call(g1));    // 1
// route 1
rule(w(a(1,Q)), g0, set(a(1,R))); // 2
rule(w(a(1,R)), g0, set(a(1,G))); // 3
// route 2
rule(w(a(2,Q)), g0, set(a(2,G))); // 4

setRulesName("g1");
rule(w(a(1,P)), g1, exit(a(1,P))); // 1
rule(w(a(1,P)), g1, set(a(2,Q)));  // 2
```

0	g0.1	-6.0509195
1	g0.2	0.0
2	g0.3	0.0
3	g0.4	-1.0
4	g1.1	-4.9427114
5	g1.2	-1.0001788

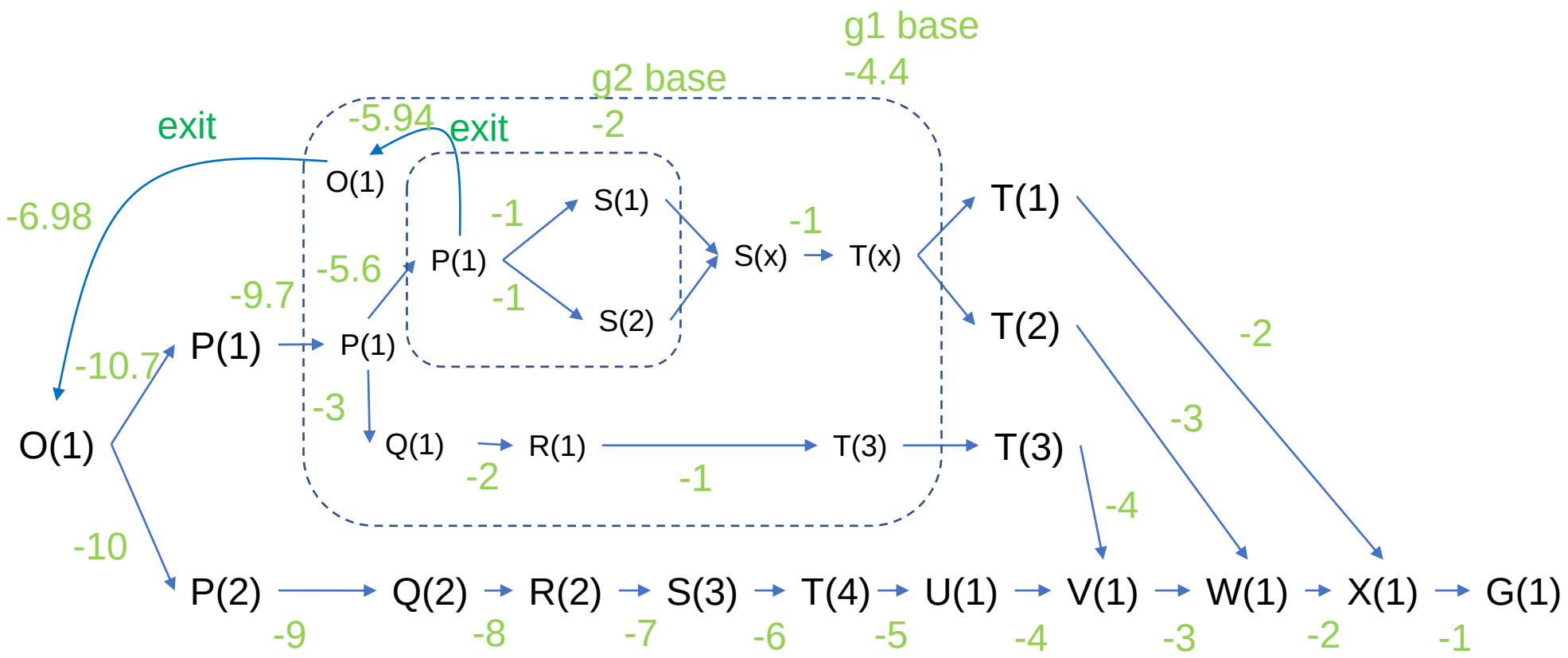
call の価値を c する
 exit の G(2) までの価値: -1 + c
 set(Q(2)) の G(2) までの価値: -3
 call(P(1)) の G(2) までの価値: -1 + (-1 + c) * 0.5 + -3 * 0.5 = c
これを解くと c = -6

テスト９ exit して再度 exit してやり直し




```
setRulesName("g0");
rule(w(a(1,O)), g0, set(a(1,P)));  // 1
rule(w(a(1,O)), g0, set(a(2,P)));  // 2
// P(1) -> call(T(+))
rule(w(a(1,P)), g0, call(g1));  // 3
// P(1) -> T(1) -> X(1)
rule(w(a(1,T)), g0, set(a(1,X)));  // 4
// P(1) -> T(2) -> W(1)
rule(w(a(2,T)), g0, set(a(1,W)));  // 5
// P(1) -> T(3) -> V(1)
rule(w(a(3,T)), g0, set(a(1,V)));  // 6
// P(2) -> Q(1) -> R(1) -> ... -> X(1) -> G(1)
rule(w(a(2,P)), g0, set(a(2,Q)));  // 7
rule(w(a(2,Q)), g0, set(a(2,R)));  // 8
rule(w(a(2,R)), g0, set(a(3,S)));  // 9
rule(w(a(3,S)), g0, set(a(4,T)));  // 10
rule(w(a(4,T)), g0, set(a(1,U)));  // 11
rule(w(a(1,U)), g0, set(a(1,V)));  // 12
rule(w(a(1,V)), g0, set(a(1,W)));  // 13
rule(w(a(1,W)), g0, set(a(1,X)));  // 14
rule(w(a(1,X)), g0, set(a(1,G)));  // 15

setRulesName("g1");
// P(1) -> S(x) -> T(x)
rule(w(a(1,P)), g1, call(g2));     // 1
rule(w(a(x,S)), g1, set(a(x,T)));  // 2
// P(1) -> Q(1) -> R(1) -> T(3)
rule(w(a(1,P)), g1, set(a(1,Q)));  // 3
rule(w(a(1,Q)), g1, set(a(1,R)));  // 4
rule(w(a(1,R)), g1, set(a(3,T)));  // 5
// P(1) -> exit(O(1)) -> exit(O(1))
rule(w(a(1,O)), g1, exit(a(1,O)));  // 6

setRulesName("g2");
rule(w(a(1,P)), g2, exit(a(1,O)));  // 1
rule(w(a(1,P)), g2, set(a(1,S)));   // 2
rule(w(a(1,P)), g2, set(a(2,S)));   // 3
```

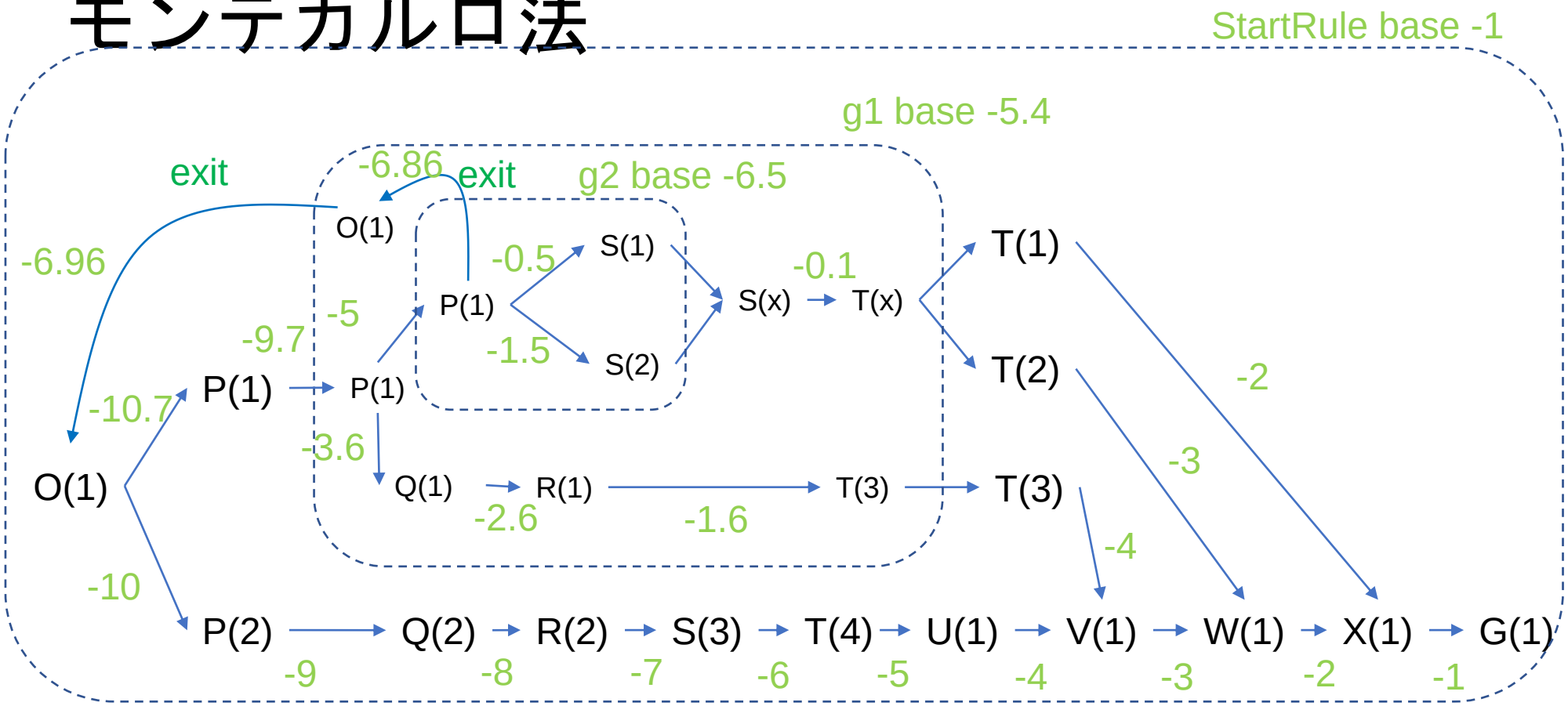
テスト９ exit して再度 exit してやり直し Sarsa



alpha = 0.001		
beta = 1e-5		
10万エピソード		
0	g0.1	-10.706895
1	g0.2	-9.9977665
2	g0.3	-9.665258
3	g0.4	-1.9997932
4	g0.5	-2.999552
5	g0.6	-3.9996724
6	g0.7	-8.998243
7	g0.8	-7.998719
8	g0.9	-6.9989576
9	g0.10	-5.999196
10	g0.11	-4.9994345
11	g0.12	-3.9996724
12	g0.13	-2.9997916
13	g0.14	-1.9999106
14	g0.15	-0.9999702
15	g1.1	-5.6036243
16	g1.2	-0.9999702
17	g1.3	-2.9997916
18	g1.4	-1.9999106
19	g1.5	-0.9999702
20	g1.6	-6.9754114
21	g2.1	-5.9433594
22	g2.2	-0.9999147
23	g2.3	-0.99988914

g1 base : (-3/6 + -4/6 + -5/2) / (5/6) = -4.4
call(g2) の価値 : -1 + -5.94 * (1/3) + -1 * (2/3) + -2 = -5.64
call(g1) の価値: -1 + -5.6 * (1/2) + -3 * (1/2) + -4.4 = -9.7
g1 から exit する価値: -1 + (-10.7 + -10)/2 - -4.4 = -6.95
g2 から exit する価値: -1 + -6.98 - -2 = -5.98

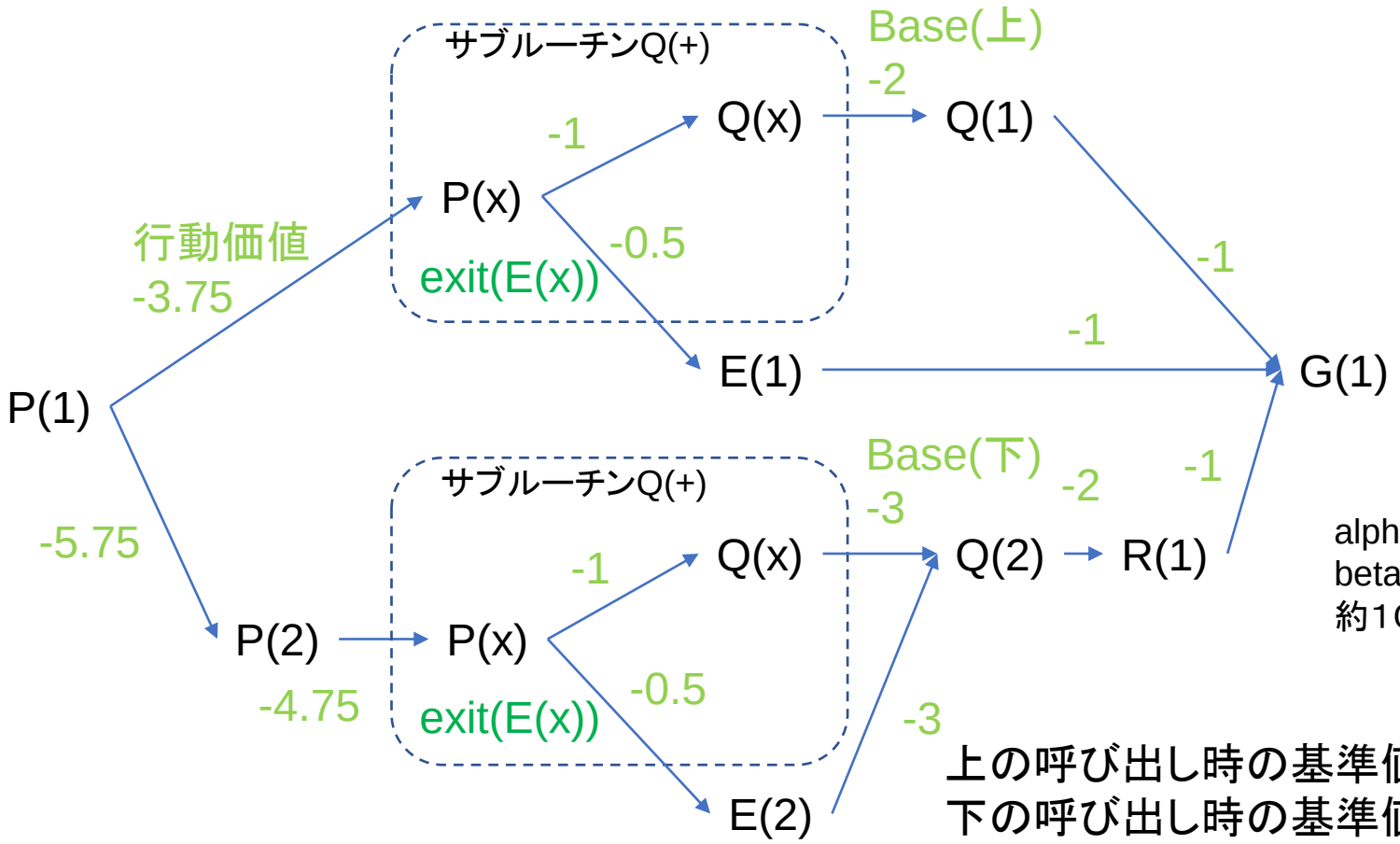
テスト９ exit して再度 exit してやり直し モンテカルロ法



alpha = 0.001		
beta = 1e-5		
10万エピソード		
0	g0.1	-10.672348
1	g0.2	-9.999554
2	g0.3	-9.672348
3	g0.4	-1.9998592
4	g0.5	-2.999698
5	g0.6	-3.9999104
6	g0.7	-8.999554
7	g0.8	-7.999791
8	g0.9	-6.999791
9	g0.10	-5.999791
10	g0.11	-4.999791
11	g0.12	-3.9999104
12	g0.13	-2.9999108
13	g0.14	-1.9999702
14	g0.15	-1.0
15	g1.1	-5.0088024
16	g1.2	-0.103944704
17	g1.3	-3.6178486
18	g1.4	-2.6178486
19	g1.5	-1.6178471
20	g1.6	-6.962229
21	g2.1	-6.8604856
22	g2.2	-0.51465994
23	g2.3	-1.5135933

g1 base : (-3/6 + -4/6 + -5/2) / (5/6) + -1 = -5.4, g2 base: -5.4 + -1 + -0.1 = -6.5
call(g2) の価値 : -1 + -6.86 * (1/3) + -1 * (2/3) + -6.5 - -5.4 = -5.05
call(g1) の価値: -1 + -5 * (1/2) + -3.6 * (1/2) + -5.4 - -1 = -9.7
g1 から exit する価値: -1 + (-10.7 + -10)/2 - -5.4 + -1 = -6.95
g2 から exit する価値: -1 + -6.96 + -5.4 - -6.5 = -6.86

テスト10 複数の呼び出し文脈 学習結果 Sarsa



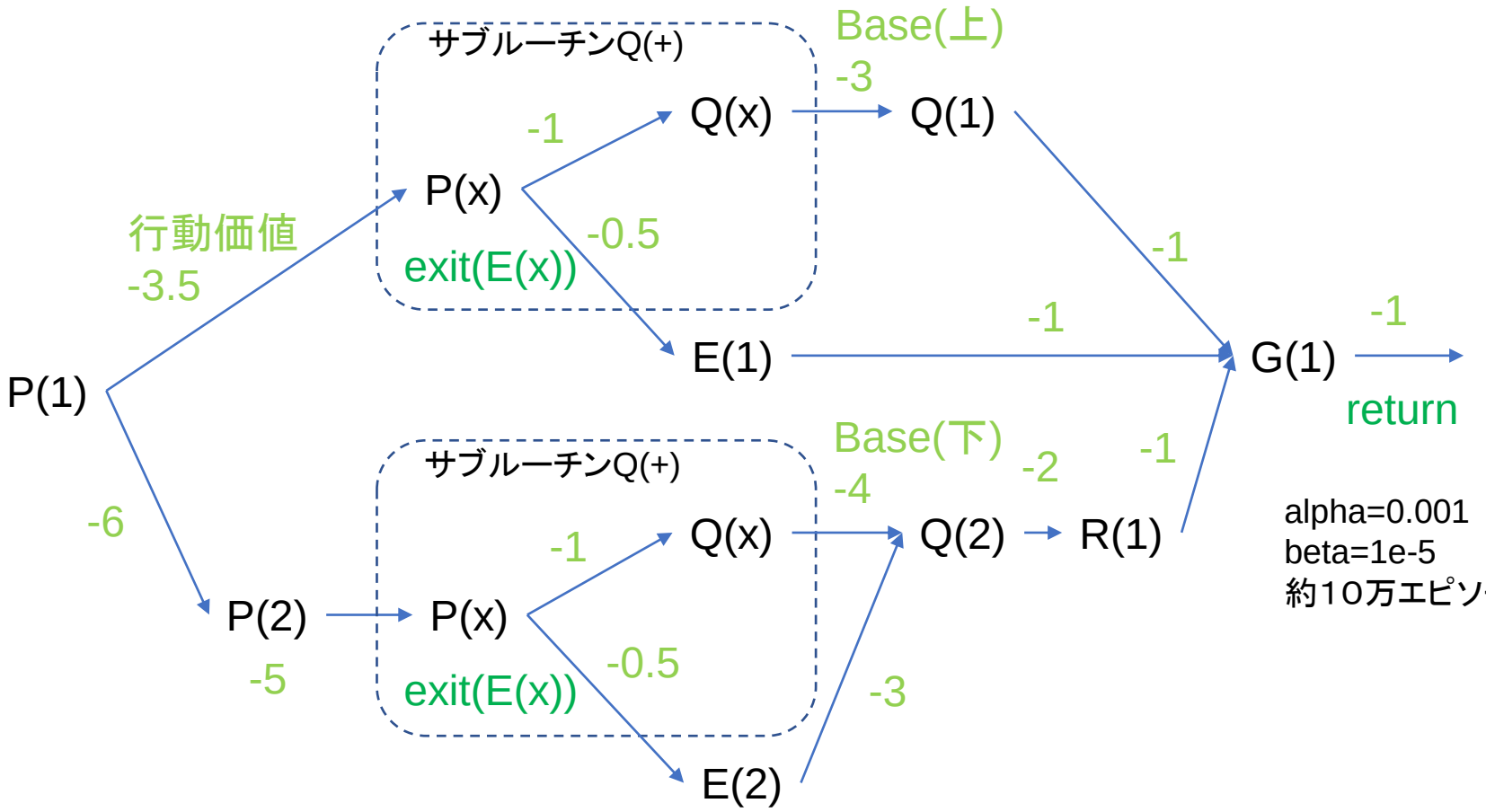

```
setRulesName("g0");
//
rule(w(a(1,P)), g0, call(g1));      // 1
rule(w(a(1,Q)), g0, set(a(1,G)));   // 2
rule(w(a(1,E)), g0, set(a(1,G)));   // 3
//
rule(w(a(1,P)), g0, set(a(2,P)));   // 4
rule(w(a(2,P)), g0, call(g1));      // 5
//   Q(2) -> R(2) -> G(1)
rule(w(a(2,Q)), g0, set(a(2,R)));   // 6
rule(w(a(2,R)), g0, set(a(1,G)));   // 7
//   E(2) -> Q(2) -> R(1) -> G(1)
rule(w(a(2,E)), g0, set(a(2,Q)));   // 8
rule(w(a(2,Q)), g0, set(a(1,R)));   // 9
rule(w(a(1,R)), g0, set(a(1,G)));   // 10

setRulesName("g1");
rule(w(a(x,P)), g1, set(a(x,Q)));   // 1
rule(w(a(x,P)), g1, exit(a(x,E)));  // 2
```

alpha=0.001
beta=1e-5
約10万エピソード学習

0	g0.1	-3.7531724
1	g0.2	-0.9999702
2	g0.3	-0.9999702
3	g0.4	-5.75141
4	g0.5	-4.754573
5	g0.6	-1.9999106
6	g0.7	-0.9999702
7	g0.8	-2.9997916
8	g0.9	-1.9999106
9	g0.10	-0.9999702
10	g1.1	-0.9999702
11	g1.2	-0.5189541

上の呼び出し時の基準値 -2
下の呼び出し時の基準値 -3
exit(E(x)) の相対価値 -1 + ((-1 - -2) + (-3 - -3) / 2 = -0.5
上の呼び出しの価値 -1 + ((-1 + -0.5) / 2 + -2) = -3.75
下の呼び出しの価値 -1 + ((-1 + -0.5) / 2 + -3) = -4.75

テスト10 複数の呼び出し文脈 モンテカルロ




```
setRulesName("g0");
//
rule(w(a(1,P)), g0, call(g1));      // 1
rule(w(a(1,Q)), g0, set(a(1,G)));   // 2
rule(w(a(1,E)), g0, set(a(1,G)));   // 3
//
rule(w(a(1,P)), g0, set(a(2,P)));   // 4
rule(w(a(2,P)), g0, call(g1));      // 5
//   Q(2) -> R(2) -> G(1)
rule(w(a(2,Q)), g0, set(a(2,R)));   // 6
rule(w(a(2,R)), g0, set(a(1,G)));   // 7
//   E(2) -> Q(2) -> R(1) -> G(1)
rule(w(a(2,E)), g0, set(a(2,Q)));   // 8
rule(w(a(2,Q)), g0, set(a(1,R)));   // 9
rule(w(a(1,R)), g0, set(a(1,G)));   // 10

setRulesName("g1");
rule(w(a(x,P)), g1, set(a(x,Q)));   // 1
rule(w(a(x,P)), g1, exit(a(x,E)));  // 2
```

alpha=0.001
beta=1e-5
約10万エピソード学習

0	g0.1	-3.496153
1	g0.2	-1.0
2	g0.3	-1.0
3	g0.4	-5.999791
4	g0.5	-4.999791
5	g0.6	-1.9999702
6	g0.7	-1.0
7	g0.8	-2.9999108
8	g0.9	-1.9999702
9	g0.10	-1.0
10	g1.1	-1.0001788
11	g1.2	-0.51904476

上の呼び出しの価値 (-4 + -3) / 2 = -3.5
下の呼び出しの価値 (-5 + -5) / 2 = -5